# Системы счисления. Перевод чисел из одной системы счисления в другую. Метод преобразования с использованием весов разрядов.

***Системы счисления***

**Система счисления** – это знаковая система, в которой числа записываются по определенным правилам с помощью символов некоторого алфавита, называемых цифрами.

Наличие различных систем счисления предполагает использование способов перевода записи числа из одной системы в другую. Для этой цели применяются следующие методы преобразований:

- преобразования с использованием весов разрядов в исходной и в искомой записи числа;

- деления (умножения) на новое основание;

- с использованием особого соотношения заданной и искомой систем счисления.

***Метод преобразования с использованием весов разрядов***

Метод имеет две разновидности в зависимости от того, какая система счисления (исходная или искомая) является более привычной. Если более привычной является искомая система, то на основании расширенной записи исходного числа подсчитываются значения ее отдельных разрядов в новой системе счисления. Далее полученные значения суммируются. Например, при преобразовании целого двоичного числа *N2*= 110011010 в десятичную систему счисления исходное число представляется в расширенной записи *N*= 28+ 27+ 24+ 23+ 21и рассчитывается вес отдельных (ненулевых) двоичных разрядов в десятичной системе счисления:

256, 128, 16, 8, 2.

Затем искомая запись числа определяется как сумма весов всех ненулевых разрядов записи числа в заданной системе счисления:

256 + 128 + 16 + 8 + 2 = 410.

При преобразовании правильных дробей в принципе используется тот же подход, но при расчете весов отдельных разрядов берутся отрицательные степени основания счисления.

*Пример*

Найти двоичный эквивалент числа 0,710=?2.

*Решение*

Предварительный результат ищется с точностью до пяти двоичных разрядов, причем пятый разряд используется только для округления при переходе к четырехразрядному окончательному результату.

Первый (старший) разрядом весом 2*–*1= 0,5 искомой двоичной записи числа будет иметь значение 1. С помощью остальных (младших) разрядов искомой записи числа необходимо представить значение 0,2 (0,2 – остаток, полученный как 0,7*–*0,5 = 0,2).

Второй (старший) разряд с весом 2*–*2= 0,25 в искомой двоичной записи числа будет иметь значение 0.

Третий разрядом с весом 2*–*3= 0,13 в искомой двоичной записи числа будет иметь значение 1. С помощью остальных (более младших) разрядов искомой записи числа необходимо представить значение 0,07 (0,07 – остаток, полученный как 0,2*-*0,13).

Четвертый разрядом с весом 2*–*4= 0,06 в искомой двоичной записи числа будет иметь значение 1, а остаток 0,01.

Пятый разряд с весом 2*–*5= 0,03 искомой двоичной записи числа будет иметь значение 0.

Таким образом, десятичное число 0,710= 0,101102.

После округления имеет место 0,710= 0,10112.

# Перевод чисел из одной системы счисления в другую. Метод деления (умножения) на новое основание.

***Метод деления (умножения) на новое основание***

**Система счисления** – это знаковая система, в которой числа записываются по определенным правилам с помощью символов некоторого алфавита, называемых цифрами.

Метод деления (умножения) имеет две разновидности соответственно для преобразования целых и дробных чисел.

1.Преобразование целых чисел

*Пример*

Найти запись в двоичной форме десятичного числа *N*10= 436.

*Решение*

Делим сначала исходное число *N*10, а затем получаемые частные на значение нового основания 2 до получения частного со значением, меньше чем 2:

436/2 → int(436/2) = 218 и rest (436/2) = 0;

218/2 → int(218/2) = 109 и rest (218/2) = 0;

109/2 →int(109/2) = 54 и rest (109/2) = 1;

54/2 → int(54/2) = 27 и rest (54/2) = 0;

27/2 → int(27/2) = 13 и rest (27/2) = 1;

13/2→int(13/2) = 6 и rest (13/2) = 1;

6/2 → int(6/2) = 3 и rest (6/2) = 0;

3/2 → int(3/2) = 1 и rest (3/2) = 1.

Таким образом: 436 = 11 0110100.

2.Преобразование дробных чисел

*Пример*

Найти запись в двоичной форме десятичного числа *M*10= 0,7.

*Решение*

Определяем количество разрядов числа *M*2. Так как исходная запись числа содержит один десятичный разряд, то запись данного числа в двоичном основании должна содержать четыре разряда. Учитывая округление, ищется предварительный двоичный эквивалент с пятью разрядами.

Умножаем исходное число *M*10, а затем дробные части последовательно получаемых произведений на новое основание 2. Выполняется пять таких операций умножения, в результате получаем:

0,7 *×*2 = 1,4 (int(0,7*×*2) = 1 и DF (0,7*×*2) = 0,4);

0,4 *×*2 = 0,8 (int(0,4*×*2) = 0 и DF(rest (0,4*×*2) = 0,8);

0,8 *×*2 = 1,6 (int(0,8*×*2) = 1 и DF(rest (0,8*×*2) = 0,6);

0,6 *×*2 =1,2 (int(0,6*×*2) = 1 и DF(rest (0,6*×*2) = 0,2);

0,2 *×*2 = 0,4 (int(0,2*×*2) = 0 и DF(rest (0,2*×*2) = 0,4).

Таким образом, 0,7 = 0,10110, а окончательный результат перехода в двоичную систему будет 0,710= 0,10112.

# Перевод чисел из одной системы счисления в другую. Метод с использованием особого соотношения оснований исходной и искомой систем счисления.

***Метод с использованием особого соотношения оснований исходной и искомой систем счисления***

Данный метод применим в тех случаях, когда исходное *q*1 и новое *q*2 основания могут быть связаны через целую степень, т.е. когда выполняются условия: *q*1^*m* = *q*2 (*условие 1*) или *q*2*^m* = *q*1 (*условие 2*). Если имеет место *условие 2*, то для заданного в системе с основанием *q*1 числа *Nq*1 *= аn аn*-1 *аn*-2... *а*1*а*0 запись его в системе в новом основании *q*2 определяется следующим образом:

- каждому разряду *ai* исходной записи числа ставится в соответствие его *m-*разрядный эквивалент в системе счисления с основанием *q*2;

- искомая запись всего заданного числа формируется за счет объединения всех полученных *m*-разрядных групп.

***Пример:***Найти двоичный эквивалент восьмеричного числа 67401.648.

*Решение*

Основания исходной и новой систем счисления можно выразить через целую степень:23 = 8.

Поэтому применяем третий метод для случая перехода из системы с большим основанием в систему с меньшим основанием. Ставим в соответствие каждой цифре исходной записи числа трехразрядный двоичный код (*триаду*):

6 7 4 0 1 6 4

110 111 100 000 001 110 100

Формируем окончательный результат посредством объединения полученных трехразрядных двоичных чисел в единый двоичный эквивалент:

67401.648 = 110111100000001.110100.

***Пример*** Найти шестнадцатеричный эквивалент двоичного числа N =11100101110110.1110110012.

*Решение*

Основания исходной и новой систем счисления можно выразить через целую степень:24=16.

Поэтому применяем третий метод для случая перехода из системы с меньшим основанием в систему с большим основанием. Разбиваем исходную запись числа на группы по четыре разряда (*тетрады*) вправо и влево от точки, в крайних левой и правой группах недостающие разряды заполняем нулями и каждой полученной группе из четырех разрядов ставим в соответствие цифру шестнадцатеричной системы счисления

0011 1001 0111 0110 . 1110 1100 1000

3 9 7 6 EC 8

Формируем окончательный результат посредством объединения полученных цифр в единый шестнадцатеричный эквивалент 11100101110110.1110110012 = 3976.EC816.

# Арифметические операции над двоичными числами. Операция сложения и вычитания в двоичной системе исчисления.

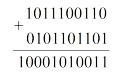
При выполнении любой операции результат ищется согласно соответствующим правилам, которые удобно представлять в табличной форме, где для всех возможных комбинаций значений одноразрядных операндов приводятся значения результата.

***Правила сложения в двоичной системе счисления***

|  |  |  |
| --- | --- | --- |
| + | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0*\** |

Все возможные значения первого слагаемого задаются во второй и третьей строках первой колонки; все возможные значения второго слагаемого – во второй и третьей колонках первой строки. На пересечении отмеченных значениями операндов строк и колонок располагается результат их сложения. В таблице знаком «*\**» отмечен случай, когда в текущем разряде результата получен ноль и имеет место перенос в ближайший старший разряд

*Пример*



***Правила вычитания в 2-й СС***

|  |  |  |
| --- | --- | --- |
| - | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1\* | 0 |

Звездочкой отмечен случай, когда в текущем разряде получена единица, путем займа из ближайшего старшего разряда.

*Пример.*

\_ 1000111001

0101101101

0011001100

В ЭВМ никогда в перечне выполняемых операций арифметического устройства не присутствует одновременно операция сложения и операция вычитания. При этом, как правило, присутствует только операции сложения. Что же касается операции вычитания, то она реализуется за счет прибавления к уменьшаемому значения вычитаемого, взятого с противоположным знаком.

# Операция умножения в двоичной системе исчисления

Возможные методы реализации операции умножения можно классифицировать по двум признакам:

– Начиная с какого разряда (со старшего или младшего) выполняется отработка множителя;

– Что сдвигается – множимое или промежуточное произведение.

Можно выделить четыре метода умножения:

– Умножение с младших разрядов множителя со сдвигом множимого;

– Умножение с младших разрядов множителя со сдвигом промежуточного произведения;

– Умножение со старшего разряда множителя со сдвигом множимого;

– Умножение со старшего разряда множителя со сдвигом промежуточного произведения;

# IEEE754. Специальные числа. Зачем нулю знак.

***IEEE754***

Разработчики «K-C-S» победили и теперь их детище воплотилось в стандарт IEEE754. Числа с плавающей запятой в нем представлены в виде знака (s), мантиссы (M) и порядка (E) следующим образом:  
  
(-1)s × 1.M × 2E  
  
*Замечание.* В новом стандарте IEE754-2008 кроме чисел с основанием 2 присутствуют числа с основанием 10, так называемые *десятичные* (decimal) числа с плавающей запятой.  
  
 В числах одинарной точности (float/single) порядок состоит из 8 бит, а мантисса – из 23. Эффективный порядок определяется как E-127. Например, число 0,15625 будет записано в памяти как  
  
  
  
В этом примере:

* Знак s=0 (положительное число)
* Порядок E=011111002-12710 = -3
* Мантисса M = 1.012 (первая единица не явная)
* В результате наше число F = 1.012e-3 = 2-3+2-5 = 0,125 + 0,03125 = 0,15625

**//Чуть более подробное объяснение**

Здесь мы имеем дело с двоичным представлением числа «101» со сдвигом запятой на несколько разрядов влево. 1,01 — это двоичное представление, означающее 1×20 + 0×2-1 + 1×2-2. Сдвинув запятую на три позиции влево получим 1,01e-3 = 1×2-3 + 0×2-4 + 1×2-5 = 1×0,125 + 0×0,0625 + 1×0,03125 = 0,125 + 0,03125 = 0,15625.

***Специальные числа: ноль, бесконечность и неопределенность***

В IEEE754 число «0» -- E=Emin-1 (для single это -127) и нулевой мантиссой. Введение нуля как самостоятельного числа (т.к. в нормализованном представлении нельзя представить ноль) позволило избежать многих странностей в арифметике. И хоть операции с нулем нужно обрабатывать отдельно, обычно они выполняются быстрее, чем с обычными числами.  
  
 Также в IEEE754 предусмотрено представление для специальных чисел, работа с которыми вызывает исключение. К таким числам относится бесконечность (±∞) и неопределенность (NaN).   
 *Неопределенность* или NaN (от not a number) – это представление, придуманное для того, чтобы арифметическая операция могла всегда вернуть какое-то не бессмысленное значение. В IEEE754 NaN представлен как число, в котором E=Emax+1, а мантисса не нулевая.  
 Как можно получить NaN? Одним из следующих способов:

* ∞+(- ∞)
* 0 × ∞
* 0/0, ∞/∞
* sqrt(x), где x<0

По определению NaN ≠ NaN, поэтому, для проверки значения переменной нужно просто сравнить ее с собой.

***Зачем нулю знак (или +0 vs -0)***  
 Так, 3·(+0)=+0, а 3·(-0)=-0. Но при сравнении +0=-0. В стандарте знак сохранили умышленно, чтобы выражения, которые в результате переполнения или потери значимости превращаются в бесконечность или в ноль, при умножении и делении все же могли представить максимально корректный результат. Например, если бы у нуля не было знака, выражение 1/(1/x)=x не выполнялось бы верно при x=±∞, так как 1/∞ и 1/-∞ равны 0.  
  
 Еще один пример:  
(+∞/0) + ∞ = +∞, тогда как (+∞/-0) +∞ = NaN

# Деление двоичных чисел (общие правила)

Деление - неточная операцией, поэтому при её выполнении прежде всего устанавливается количество разрядов частного, которые подлежат определению. Деление в двоичной системе счисления может выполняться точно так же, как и в десятичной, однако формирования частного двоичных операндов реализуется гораздо проще, чем в десятичной системе, т.к.:

• Упрощается процедура подбора очередной цифры вследствие того, что в двоичной системе очередной цифрой может быть одна из двух - либо 0, либо 1;

• Упрощается процедура умножения найденной цифры частного на делитель.

# Деление двоичных чисел с восстановлением остатка

***Алгоритм деления с восстановлением остатка*** состоит в следующем.

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

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

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

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

5. Окончание алгоритма.

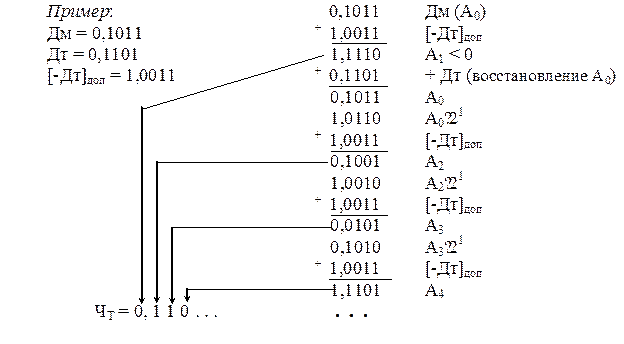
Из рассмотренного алгоритма видно следующее:

1) необходимо затрачивать время на восстановление остатка;

2) процесс деления нерегулярный, в зависимости от делимого и делителя

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

Рассмотрим пример деления чисел.



# Деление двоичных чисел без восстановления остатка.

***Алгоритм деления без восстановления остатка***

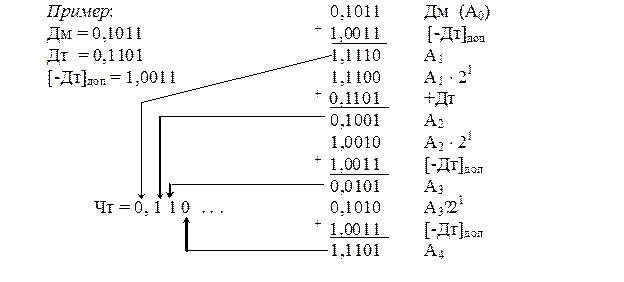
1. Выполняется пробное вычитание с формированием первого остатка A1=[Дм]доп+[-Дт]доп. Далее, если А1 < 0, то в первый разряд, расположенный слева от запятой, заносится ноль (0), иначе единица (1) – что является признаком переполнение и осуществляется переход к пункту 5.

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

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

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

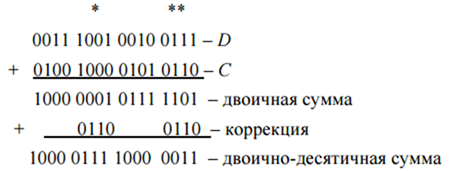
5. Окончание алгоритма.



# Двоично-десятичная арифметика. Сложение и вычитание двоично-десятичных чисел.

***Сложение двоично-десятичных чисел***

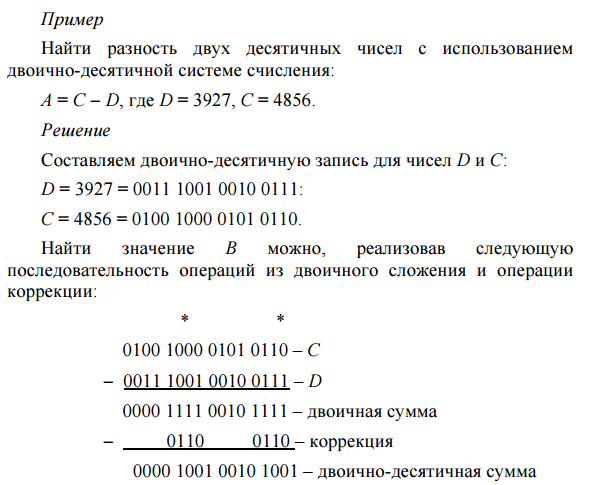
Рассмотрим на конкретном примере реализацию этой операции. Пример Найти сумму двух десятичных чисел с использованием двоично-десятичной системы счисления: A = D + C, где D = 3927; C = 4856. Решение Составляем двоично-десятичную запись для чисел D и C: D =3927 = 0011 1001 0010 0111: C = 4856 =0100 1000 0101 0110. Найти значение А можно, реализовав следующую последовательность операций из двоичного сложения и операции коррекции:



Для получения двоично-десятичной суммы A на основании результата сложения операндов по правилам двоичной арифметики необходимо добавить шестерку (0110) в те тетрады, из которых был перенос. В данном примере это вторая тетрада (отмечена \*). Необходимость такой коррекции обусловливается тем, что перенос, сформированный по правилам двоичного суммирования, унес из тетрады шестнадцать, а для десятичного сложения перенос должен был унести десять, т.е. перенос, сформированный по правилам двоичной арифметики, унес лишнюю шестерку. Кроме этого шестерка добавляется в те тетрады, в которых получено значение, большее девяти. Такая коррекция обуславливается тем, что по правилам десятичной арифметики в 25 таких тетрадах должен быть выработан перенос и, чтобы его выработать по правилам двоичной арифметики, в тетраду нужно добавить шестерку. Для рассмотренного примера такой тетрадой является и четвертая тетрада (отмечена \*\*)

Вычитание двоично-десятичных чисел

В ЭВМ часто предусматривается обработка чисел не только в двоичной системе счисления, но в двоично-десятичной. При этом, как правило, стремятся реализовать двоично-десятичную арифметику по правилам двоичной с введением ограниченного количества коррекций.



# Кодирование алгебраических чисел. Дополнительный и обратный коды двоичных чисел.

***Кодирование алгебраических чисел***

Для представления чисел со знаком используются специальные коды:

- прямой код;

- дополнительный код;

- обратный код.

Во всех трёх случаях используется следующий формат представления числа, содержащий два поля - поле знака и поле модуля

*Поле знака* представлено одним разрядом, в котором устанавливается 0, если число положительное, и 1, если число отрицательно.

*Поле модуля* отражает количественную оценку числа и для каждого кода формируется по*–*разному. Количество разрядов поля модуля определяется диапазоном изменения отображаемых чисел или точностью их представления.

В *прямом (1) и дополнительном (2) коде* запись целого числа*А* формируется по следующим правилам:

(1)https://studfile.net/html/20260/740/html_gatsd6GBaT.XiuA/img-C2EBxE.png(2)https://studfile.net/html/20260/740/html_gatsd6GBaT.XiuA/img-O6LW5C.png

где *n*– разрядность модульного поля;

*q*– основание системы счисления;

*qn*– максимальная не включенная граница диапазона изменения представляемых чисел, т. к. диапазон изменение чисел *А* определяется как *qn*> |*А*| ≥ 0 .

Для случая правильной дроби запись числа *А* в дополнительном коде имеет вид:

https://studfile.net/html/20260/740/html_gatsd6GBaT.XiuA/img-I5_21A.png

где 1 – максимальная невключенная граница диапазона изменения представляемых чисел, т. е. диапазон изменение чисел *А* определяется как 1 > |*A* | ≥ 0.

В *обратном коде* запись целого числа *А* формируется по следующему правилу:

https://studfile.net/html/20260/740/html_gatsd6GBaT.XiuA/img-FnMl_M.png

где *n*– разрядность модульного поля;

*q*– основание системы счисления;

(2*n-*1) – максимальная включенная граница диапазона изменения представляемых чисел, т. е. диапазон изменения чисел *А* определяется как

(*qn–*1) ≥ | *A* | ≥ 0.

Для случая правильной дроби запись числа *А* в обратном коде имеет вид

https://studfile.net/html/20260/740/html_gatsd6GBaT.XiuA/img-3YqYQD.png

где *n –*разрядность поля модуля;

1 *- q-n*– верхняя включенная граница представляемых чисел.

Т. о., диапазон изменения чисел*А* определяется с. о.:

(1 *– q-n*) ≥ | *А*| ≥ 0.

Легко показать, что перевод отрицательного числа из обратного или дополнительного кода в прямой выполняется по тем же правилам, что и перевод числа из прямого кода в обратный или *дополнительный*:

– для перевода отрицательного числа из обратного в *прямой код* необходимо дополнить его модуль до включенной границы;

– для перевода отрицательного числа из обратного в *прямой код* необходимо дополнить его модуль до невключенной границы.

# Операции с двоичными числами в дополнительном и обратном кодах.

***Операции с двоичными числами в дополнительном коде***

При использовании дополнительного или обратного кода операция вычитания заменяется операцией сложения с изменением знака второго операнда. При сложении чисел, представленных в дополнительном коде, выполняется сложение разрядов по правилам двоичной арифметики по всей длине записи чисел, не обращая внимание на границу, разделяющую знаковое и модульные поля. Переполнение знакового поля игнорируется!

Исходя из абсолютного значения операндов, разрядность представления модульной части n должна быть равна количеству разрядов большего операнда. Учитывай то, что мы используем две операции: сложение и вычитание, поэтому возможно переполнение из-за переноса из старшего разряда, БЕРИ длину модульной части на один разряд больше, т.е n+1

***Операции с двоичными числами в обратном коде***

При сложении чисел, представленных в *обратном* коде, выполняется сложение разрядов, представляющих запись операндов, по правилам двоичной арифметики по всей длине записи чисел, не обращая внимания на границу, разделяющую знаковое и модульные поля. Переполнение знакового поля, т.е. перенос, возникший из крайнего левого разряда, должен быть учтен как +1 в младший разряд полученной суммы. В результате такого сложения будет получен *обратный* код суммы заданных операндов.

**Пример**

Найти значения для *С*1, *С*2, *С*3, *С*4, определяемых выражениями

*С*1 = *А*+ *В*, *С*2 = *А*- *В*, *С*3 = *В*- *А*, *С*4 = - *А*- *В*,

если А=5710, В = - 21010. При выполнении операций использовать двоичный обратный код. Результат представить в прямом коде.

**Решение**

В данном примере используются те же выражения и те же операнды, что и в предыдущем примере, поэтому при его решение используются уже найденные ранее двоичные представления операндов и их прямые коды.

Обратные коды операндов имеют вид

[*А*]ок = 0.000111001, [-*А*]ок = 1.111000110,

[*В*]ок = 1.100101101, [-*В*]ок = 0.011010010.

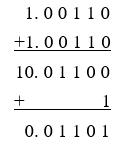
Используя сформированный дополнительный код, реализуем выражения для *С*1, *С*2, *С*3, *С*4.



В данном случае также возникло переполнение знакового разряда, которое должно быть учтено как +1 в младший разряд сформированной суммы

# Модифицированные коды

Рассмотрим пример:



Пример, выполненный по всем формальным правилам, дал абсурдный результат, так как получена положительная сумма двух отрицательных операндов. Аналогичная ситуация может возникать и при использовании дополнительного кода.

Ситуацию переполнения можно обнаруживать по факту появления «абсурдного» результата, но для этого необходимо помнить то, что в суммировании принимают участие операнды с одинаковыми знаками и знак полученного при этом результата отличен от знака операндов.

Более просто ситуация переполнения определяется при применении модифицированного кода (обратного или дополнительного). Модифицированные коды отличаются от базовых кодов только тем, что поле знака операндов имеет два разряда, и эти разряды имеют одинаковые значения:

00 – для положительных чисел;

11 – для отрицательных чисел.

# Логические операции с двоичными кодами: логическое суммирование, логическое умножение, логическое отрицание, суммирование по модулю два, логические сдвиги.

***Логические операции с двоичными кодами***

Над двоичными кодами могут выполняться различные логические операции, среди которых особое место занимают:

1) *логическое суммирование* (обозначения – ИЛИ, ОR, «∨»);

2) *логическое умножение*(обозначения – И, AND, «∧»);

3) *отрицание*(обозначения – НЕТ, NOT, «*x*», т.е. штрих над отрицаемым *x*);

4) *суммирование по модулю* 2 (обозначается mod 2, « ⊕ »);

5) *операции сдвига*.

**Логическое суммирование**

Операция *логического суммирования* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если хотя бы в одном операнде в *i*-м разряде имеет место единица.

*Пример:*

10001101 ∨ 11110000 = 11111101.

**Логическое умножение**

Операция *логического умножения* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если оба операнда в этом *i*-м разряде имеются единицу, и ноль во всех других случаях.

*Пример*:

10001101 ∧ 11110000 = 10000000

**Суммирование по модулю 2**

Операция *суммирования по модулю 2* выполняется над двумя кодами и генерирует код той же разрядности, что и операнды, у которого в некотором *i*-м разряде находится единица, если два заданных операнда в *i*-м разряде имеют противоположные значения. Иногда эта операция называется «исключающее ИЛИ».

*Пример:*

10001101 ⊕ 11110000 = 01111101.

**Логическое отрицание**

Операция логического отрицания выполняется над одним кодом и генерирует результирующий код той же разрядности, что и операнд, в некотором *i*-м разряде которого находится значение, противоположное значению в *i*-м разряде отрицаемого кода.

**Операция сдвига**

Операции *сдвига* в свою очередь, подразделяются на:

1) логические сдвиги, которые имеют разновидности – сдвиг вправо, сдвиг влево, циклический сдвиг вправо, циклический сдвиг влево;

2) арифметические сдвиги вправо и влево, выполнение которых зависит от знака и кода сдвигаемого числа.

# Арифметические сдвиги положительных двоичных чисел, представленных в прямом коде. Арифметические сдвиги двоичных чисел, представленных в обратном коде.

***Арифметические сдвиги положительных двоичных чисел, представленных в прямом коде***

Сдвиг, при котором уходящий бит исчезает, не влияя на оставшиеся биты, а на месте появившегося бита записывается бит **0**. Арифметические сдвиги обеспечивают выполнение умножения (сдвиги влево) или операции деления (сдвиги вправо) двоичных кодов на два, точно так же, как сдвиги вправо и влево десятичного числа обеспечиваю выполнение деления и умножение на 10. Если сдвигается положительное число, то сдвиг (вправо или влево) выполняется как соответствующий логический сдвиг (влево или вправо), с той лишь разницей, что предусматриваются средства определения факта переполнения при сдвиге влево, что реализуется и при всех других арифметических операциях. При любом сдвиге вправо предусматриваются средства для округления после завершения нужного количества сдвигов и средства обнаружения обнуления сдвигаемой величины после очередного сдвига. Арифметические сдвиги влево положительных двоичных чисел выполняются независимо от используемого кода (прямого, обратного, дополнительного).

Пример: Найти результат арифметического сдвига влево на три разряда двоичного прямого кода числа [А]пк= 00.00000101

Решение: Процесс выполнения заданного сдвига дает следующие промежуточные и конечное значения:

первый сдвиг: 00.00000101 ← 00.00001010;

второй сдвиг: 00.00001010 ← 00.00010100;

третий сдвиг: 00.00010100 ← 00.00101000.

***Арифметические сдвиги отрицательных двоичных чисел, представленных в прямом коде***

Арифметические сдвиги влево и вправо реализуются по-разному в зависимости как от знака числа, так и от используемого кода (прямого обратного, дополнительного).

При арифметическом сдвиге отрицательного двоичного числа, представленного в прямом коде, осуществляется соответствующий сдвиг только модульного поля записи числа.

Реализация этого типа сдвига иллюстрируется следующими примерами.

**Пример 1**

Выполнить арифметический сдвиг влево двоичного числа А = 11. 001010 (соответствует 1010), представленного в модифицированном прямом коде.

**Решение**

Заданный сдвиг, имеющий своей целью получение результата, в два раза превышающего по абсолютному значению значение исходного кода, дает в результате 11.010100 (2010), которое получается за счет логического сдвига влево только модульной части исходного кода.

Факт получения переполнения устанавливается по наличию единичного значения старшего разряда в сдвигаемом коде перед очередным сдвигом.

***Арифметические сдвиги двоичных чисел, представленных в обратном коде***

При арифметическом сдвиге влево отрицательного двоичного числа, представленного в обратном коде, осуществляется циклический сдвиг исходного кода с контролем за переполнением, например, сдвиг влево отрицательного двоичного числа 11.1100110 (2510), представленного в обратном коде, дает в результате 11.1001101 (5010).

При арифметическом сдвиге вправо отрицательного двоичного числа, представленного в обратном коде, осуществляется сдвиг только модульной части записи числа с установкой единицы в освобождающийся разряд. При этом может осуществляется контроль за обнулением результата сдвига (появление единичных значений во всех разрядах) и округление результата после выполнения заданного количества сдвигов.

*Пример3*

Выполнить сдвиг вправо на четыре разряда двоичного числа 11.1001101 (десятичный эквивалент – 5010), представленного в обратном коде.

Первый сдвиг дает 11.11001101 (5010) à 11.11100110 (2510).

Второй сдвиг дает 11.11100110 (2510) à 11.11110011 (1210).

Третий сдвиг дает 11.11110011 (1210) à 11.11111001 (610).

Четвертый сдвиг дает 11.11111001 (610) à 11.11111100 (310).

При выполнении сдвига вправо нечетного числа результат получается с точностью до младшего разряда кода, причем ошибка отрицательная.

После выполнения последнего, четвертого сдвига выполняется округление, при котором, если последний «вытолкнутый» разряд имел значение 0, к результату последнего сдвига прибавляется – 1.

Данное округление можно выполнить за счет прибавления единицы к прямому коду, соответствующему результату последнего сдвига исходного обратного кода.

В рассмотренном примере корректировать на единицу результат четвертого сдвига не надо, так как «вытолкнутый» разряд при последнем (четвертом) сдвиге равен единице. В данном случае конечный результат сдвига заданного отрицательного числа, представленного в обратном коде, равен 11.11111100.

# Арифметические сдвиги двоичных чисел, представленных в дополнительном коде. Сдвиг отрицательных чисел с переполнением.

При арифметическом сдвиге влево отрицательного двоичного числа, представленного в дополнительном коде, осуществляется логический сдвиг влево модуля исходного кода (освобождающийся разряд заполняется нулем) с контролем за переполнением, например, сдвиг влево отрицательного двоичного числа 11.11001110 (50 в 10 с/c), представленного в дополнительном коде, дает в результате 11.10011100 (100 в 10 с/с).

При арифметическом сдвиге вправо отрицательного двоичного числа, представленного в дополнительном коде, осуществляется логический сдвиг вправо модуля записи числа с установкой единицы в освобождающийся разряд. При этом может осуществляется контроль за обнулением результата сдвига (появление единичных значений во всех разрядах).

**Пример**

Выполнить сдвиг вправо на четыре разряда двоичного числа 11.11001110 (десятичный эквивалент – 50 в 10 с/c), представленного в дополнительном коде.

Решение

Первый сдвиг дает 11.11001110 → 11.11100111 (25 в 10 с/c).

Второй сдвиг дает 11.11100111 → 11.11110011 (13 в 10 с/c).

Третий сдвиг дает 11.11110011 → 11.11111001 (7 в 10 с/c).

Четвертый сдвиг дает 11.11111001 → 11.11111100 (4 в 10 с/c).

При выполнении сдвига вправо нечетного целого числа результат получается с точностью до младшего разряда кода, причем ошибка положительная.

Арифметический сдвиг вправо может выполняться над отрицательными числами с переполнением (такие числа в модифицированном прямом, обратном или дополнительном коде имеют в знаковом поле 10). В этом случае после сдвига в знаковом поле будет 11, а в старшем разряде – 0, если число представлено в обратном или дополнительном коде, или 1, если число представлено в прямом коде.

**Пример1**

Выполнить сдвиг вправо на 2 разряда числа [А]пк = 10.01000110 (А10 = 326).

**Решение**

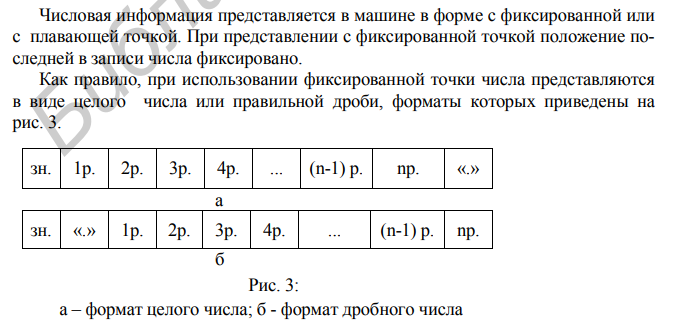
1-й сдвиг: 10. 01000110 → 11.10100011 (-163 в 10 с/c);

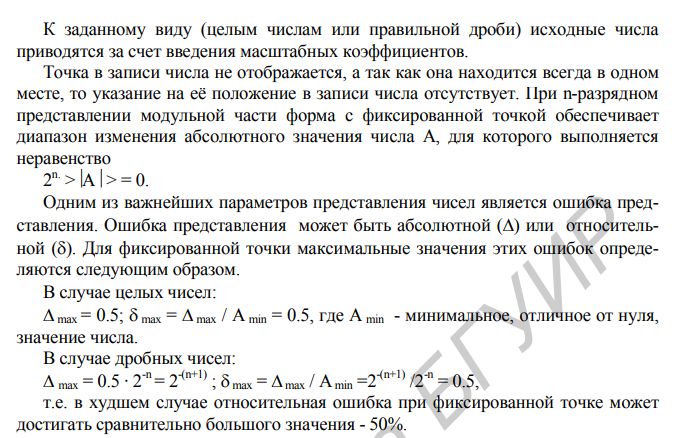
2-й сдвиг: 11.10100011 → 11.11010001 (-81 в 10 с/c) и последний вы-толкнутый разряд равен 1).

С учетом округления имеем окончательный результат:

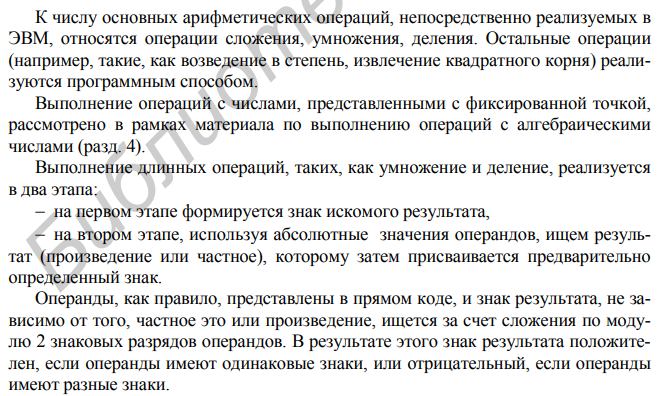
[А2]пк=11.100100101

# Представление чисел с фиксированной точкой. Арифметические операции над числами, представленными с фиксированной точкой.

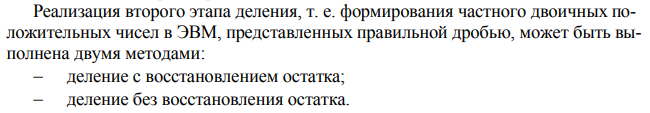
***Представление чисел с фиксированной точкой***

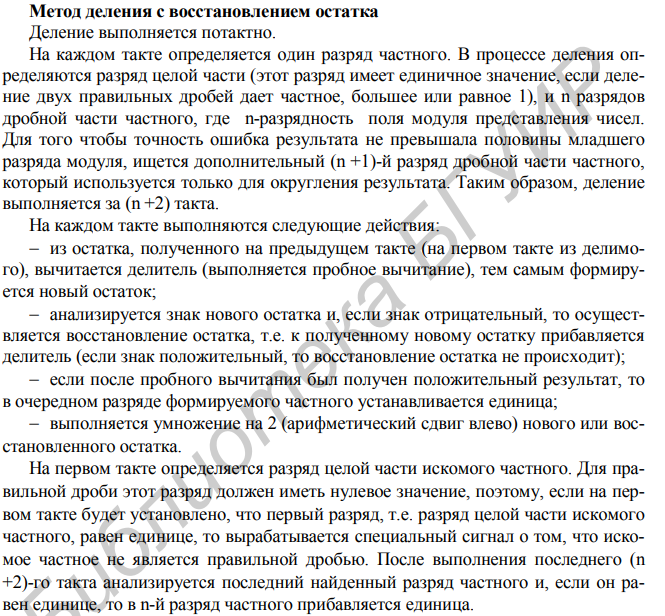


***Арифметические операции над числами, представленными с фиксированной точкой***

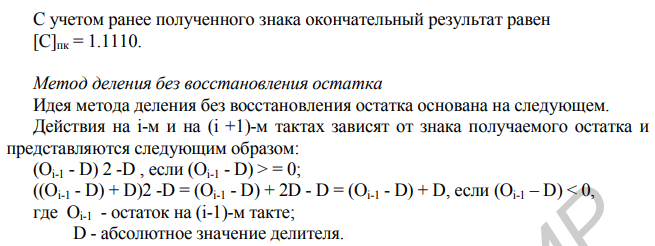


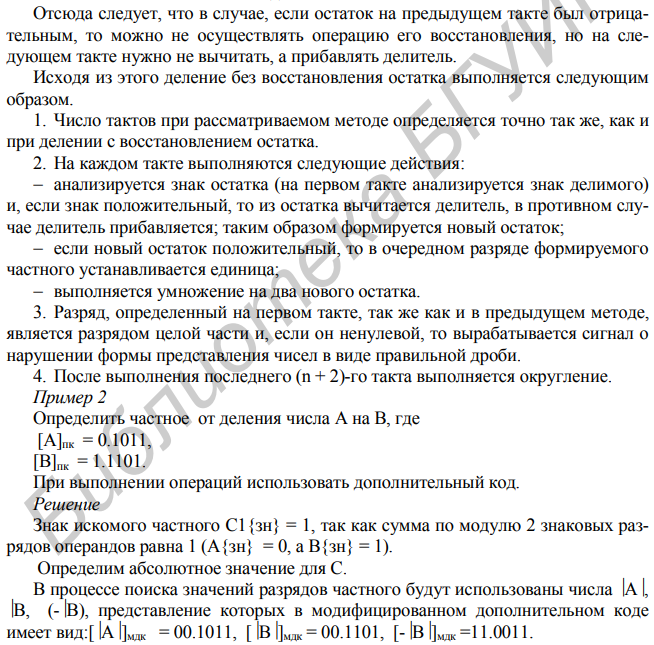
***Деление с фиксированной точкой***



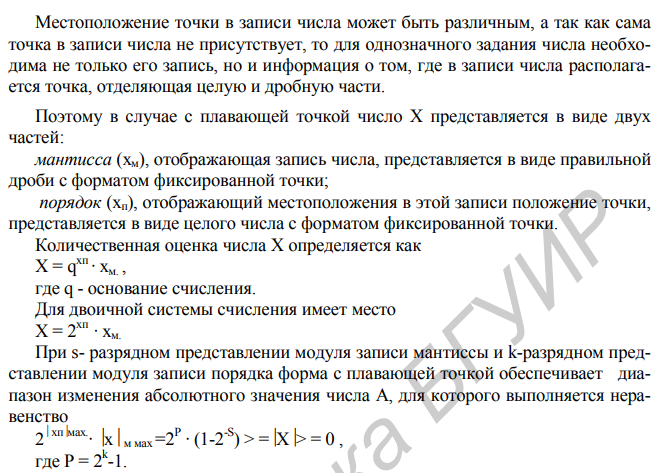


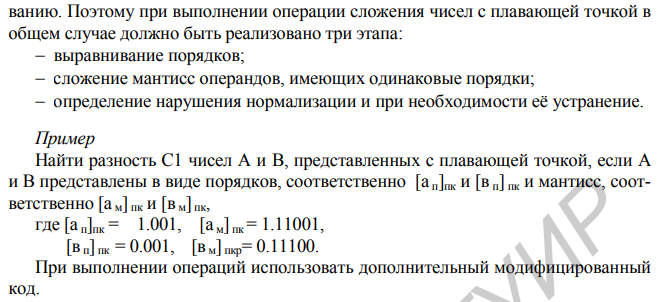






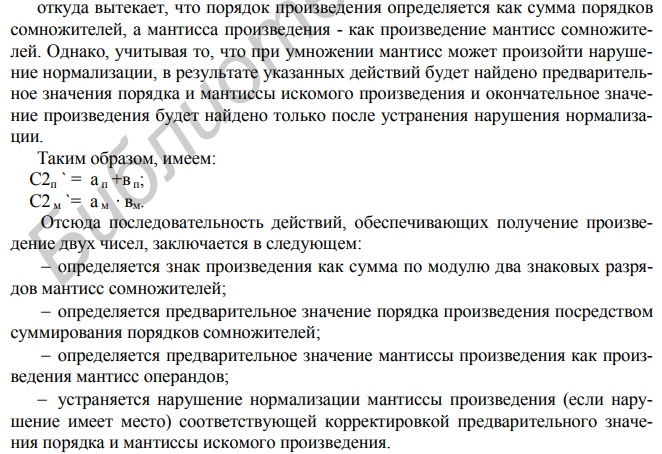
# Представление чисел с плавающей точкой. Сложение чисел, представленных в формате с плавающей точкой

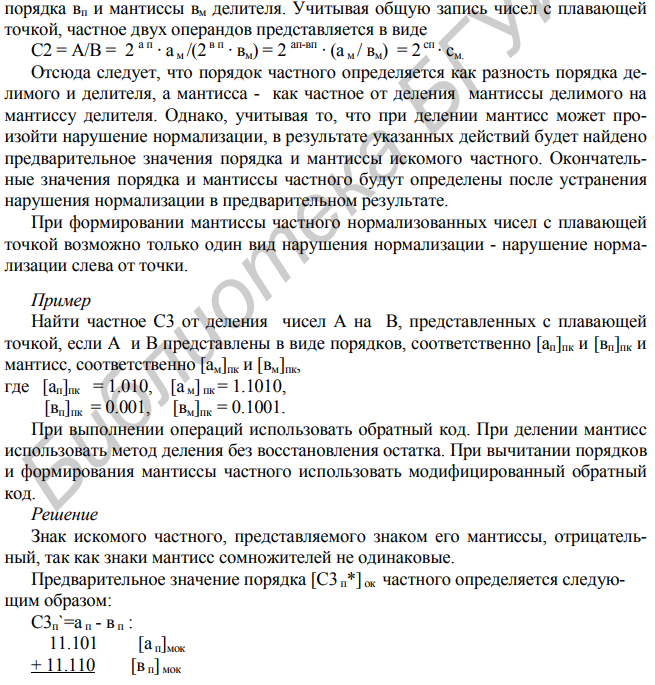
***Представление чисел с плавающей точкой***

***Сложение чисел, представленных в формате с плавающей точкой***

# Умножение чисел, представленных в формате с плавающей точкой. Деление чисел, представленных в формате с плавающей точкой.

***Умножение***



***Деление***

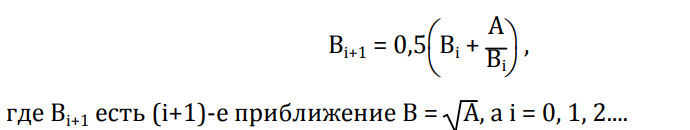
# Неосновные арифметические операции. Вычисление квадратного корня

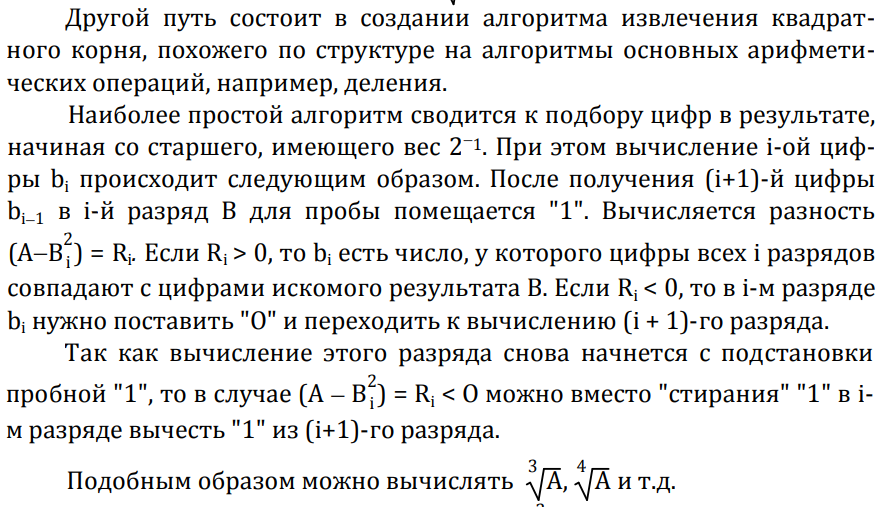
***Неосновные арифметические операции***

Неосновные арифметические операции, такие как извлечение квадратного корня, вычисление тригонометрический и других элементарных выражений, реализуются обычно с помощью стандартных программ, которые входят в состав математического обеспечения (МО) и вызываются простым обращением к соответствующей библиотеке подпрограмм. Однако для реализации этих подпрограмм требуется значительно больше времени, чем для выполнения основных арифметических операций.

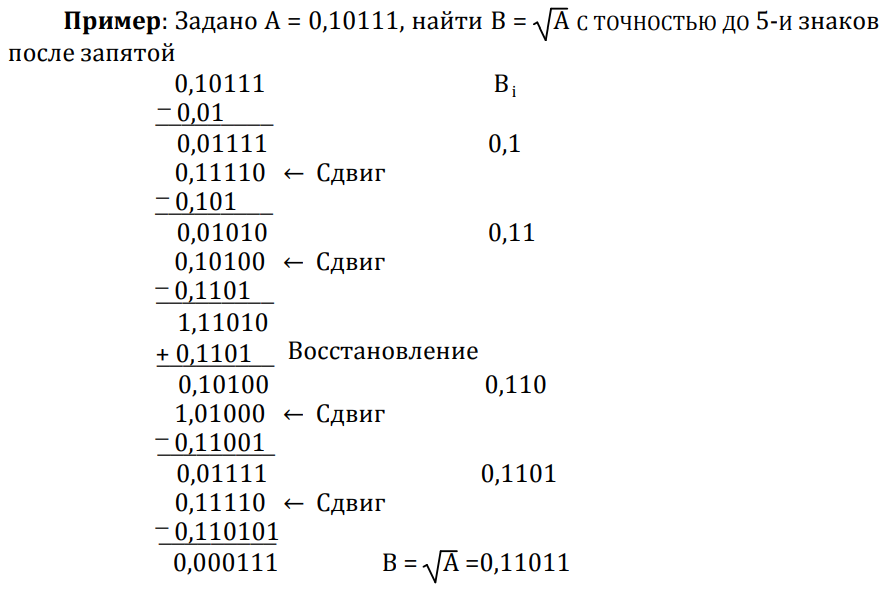
Вычисление квадратного корня

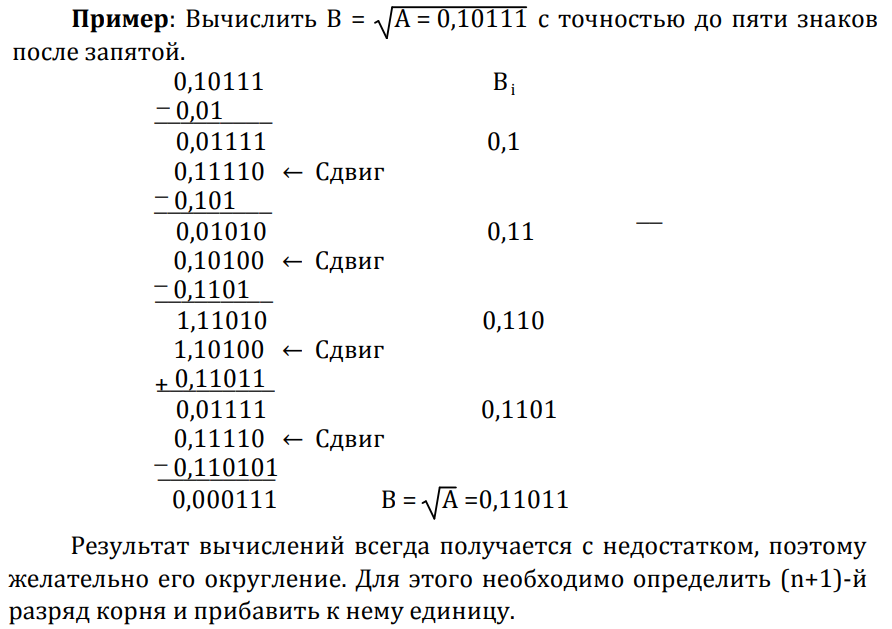
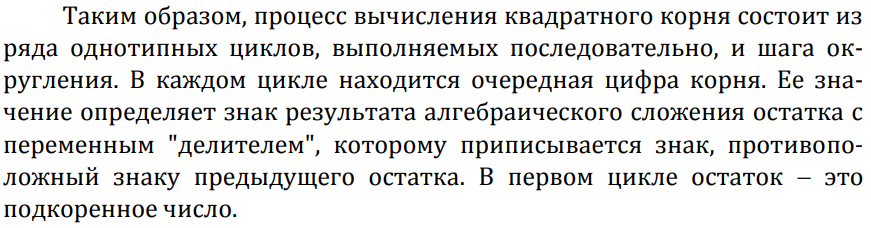
Есть два пути решения задачи. Первый путь связан с разработкой программы извлечения квадратного корня с использованием набора арифметических команд. При этом программа реализует итерационный метод извлечения квадратного корня. Например, в универсальных ЭВМ для вычисления квадратного корня применяется формула Ньютона:











# Методы вычисления элементарных функций.

В цифровой вычислительной технике применяют следующие методы вычисления элементарных функций (ЭФ):

1) Разделение в ряд Тейлора (степенные полиномы),

2) Аппроксимацию с помощью различных полиномов,

3) Табличные методы,

4) Рациональные приближения ЭФ,

5) Использование цепных дробей,

6) Итерационные (рекуррентные) методы.

Степенные полиномы (отрезок ряда Тейлора, полином Чебышева и т.д.) вычисляются в ЭВМ чаще всего по схеме Горнера. При этом требуется выполнить m операций умножения и m операций сложения (m − степень полинома). Недостатки: ряд Тейлора очень медленно сходится для некоторых функций (натуральный логарифм, обратные тригонометрические и гиперболические функции), и поэтому время вычисления будет большим, а инструментальная погрешность увеличивается. Достоинством разложения в ряд Тейлора (в отличие от аппроксимации полиномом Чебышева) является то, что можно вычислять коэффициенты членов ряда непосредственно при вычислении функций и не хранить их в памяти ЭВМ.

Метод полиномиальной аппроксимации используется в ЭВМ наиболее часто. Он характеризуется достаточно высоким единообразием вычисления всех ЭФ, однако при этом в памяти необходимо хранить большое количество коэффициентов всех полиномов. Для ускорения сходимости полинома аргумент предварительно сводится в более узкую область. Методическая погрешность знакопеременна и равномерно распределена на интервале изменения аргумента. Для вычисления ЭФ с произвольной разрядностью в некоторых ЭВМ используется комбинированный таблично-полиномиальный алгоритм. Приближение любой ЭФ в приведенном интервале ведется с помощью подпрограмм не одним ортогональным полиномом, а их набором, каждый из которых применяется на подинтервалах с возрастанием степени аппроксимации от одного подинтервала к следующему.

Табличные методы основаны, главным образом, на кусочно-линейной и криволинейной аппроксимации. Для вычисления ЭФ этим методом требуется выполнить малое число арифметических операций, однако объем таблиц и время поиска в них может быть большим. Поэтому этот метод применяется в машинах с небольшой разрядностью слов.

При методе рационального приближения ЭФ функцию представляют в виде отношения двух полиномов, причем число членов в каждом полиноме намного меньше, чем при соответствующем разложении в ряд Тейлора. Однако коэффициенты полиномов должны обязательно храниться в памяти. Для вычисления ЭФ следует вычислить два полинома и выполнить операцию деления. Надо проверять устойчивость метода.

Метод цепных дробей характеризуется однообразием вычисления всех ЭФ, при этом количество констант мало. По сравнению с методом степенных полиномов в этом методе число шагов меньше, однако на каждом шаге выполняется операция деления, занимающая много времени. Хотя для цепных дробей область сходимости более широкая, чем для ряда Тейлора, но с ростом аргумента резко возрастает необходимое число звеньев дроби. Это заставляет приводить аргументы к интервалу, не более широкому, чем при разложении в ряд Тейлора. Оценка погрешности является сложной. Метод используется в малых машинах, где быстродействие не очень важно. Например, в машине "Проминь" метод цепных дробей реализован микропрограммно.

Итерационные методы предполагают вычисление последовательных приближений функции по итерационной формуле Yj+i = f(Yi). Необходимость в ряде случаев вычислять операции деления и умножения на каждой итерации уменьшает быстродействие вычислений. Оценка погрешности удобна, алгоритмы вычисления некоторых функций (базового набора) достаточно единообразны. Загрузка памяти наименьшая, так как константы можно вычислить непосредственно перед счетом функций по той же схеме. Это снижает быстродействие, но является решающим преимуществом при использовании в машинах с произвольной разрядностью.

Для большинства других методов характерно отсутствие единообразной методики вычисления всех ЭФ. Это приводит к тому, что выбирается набор, так называемых, базовых функций, вычисляемых выбранным методом, а остальные ЭФ выражаются через базовые и вычисляются на их основе. Например, в машине МИР базовый набор ЭФ состоит из функций ln x, ax, cosx, arcsinx. Такая методика приводит к многоуровневой организации управления, усложнению структуры управляющего устройства и к увеличению времени вычисления.

В последнее время разработаны более эффективные итерационные алгоритмы вычисления ЭФ. Этот метод чаще всего называется методом "цифра за цифрой", так как после n итераций алгоритма получается значение функции с точностью до единицы n-го разряда.

Таким образом, метод "цифра за цифрой" обладает следующими преимуществами при структурной реализации: высокое быстродействие алгоритмов, основанных на операциях сдвига и сложения; единообразие вычисления почти всех ЭФ и оправданные аппаратурные затраты; простая организация вычислительного процесса, малое число уровней управления; удобство аппаратной компенсации погрешностей, возникающих при реализации алгоритмов на ЭВМ.

# Денормализованные числа. Подводные камни в арифметике с плавающей запятой.

***Денормализованные числа***

Мы знаем, что при E=Emin-1 (для float это «-127») и нулевой мантиссе число считается равным нулю. Если же мантисса не нулевая, то число считается не нулевым, его порядок полагается E=Emin, причем неявный старший бит мантиссы полагается равным нулю. Такие числа называются *денормализованными*.  
  
Строго говоря, числа с плавающей запятой теперь имеют вид:  
  
(-1)s × 1.M × 2E, если Emin≤E≤Emax (нормализованные числа)  
  
(-1)s × 0.M × 2Emin, если E=Emin-1. (денормализованные числа)

***Подводные камни в арифметике с плавающей запятой***

1. **Округление**

Особенности округления в IEEE754:

* Округление до ближайшего в стандарте сделано не так как мы привыкли. Математически показано, что если 0,5 округлять до 1 (в большую сторону), то существует набор операций, при которых ошибка округления будет возрастать до бесконечности. Поэтому в IEEE754 применяется правило округления до четного. Так, 12,5 будет округлено до 12, а 13,5 – до 14.
* Самая опасная операция с точки зрения округления в арифметике с плавающей запятой — это вычитание. При вычитании близких чисел значимые разряды могут потеряться, что  
  может в разы увеличить относительную погрешность.
* Для многих широко распространенных математических формул математики разработали специальную форму, которая позволяет значительно уменьшить погрешность при округлении. Например, расчет формулы «x2-y2» лучше вычислять используя формулу «(x-y)(x+y)».

**2) Неассоциативность арифметических операций**В арифметике с плавающей запятой правило (a\*b)\*c = a\*(b\*c) не выполняется для любых арифметических операций. Например,  
  
(1020+1)-1020=0 ≠ (1020-1020)+1=1  
  
**3)Числовые константы**

Помните, что не все десятичные числа имеют двоичное представление с плавающей запятой. Например, число «0,2» будет представлено как «0,200000003» в одинарной точности. Соответственно, «0,2 + 0,2 ≈ 0,4».

1. **Выбор минимума из двух значений**

Допустим из двух значений нам нужно выбрать минимальное. В Си это можно сделать одним из следующих способов:

1. x < y? x: y
2. x <= y? x: y
3. x > y? y: x
4. x >= y? y: x

Часто компилятор считает их эквивалентными и всегда использует первый вариант, так как он выполняется за одну инструкцию процессора. Но если мы учтем ±0 и NaN, эти операции никак не эквивалентны:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | y | x < y? x: y | x <= y? x: y | x > y? y: x | x >= y? y: x |
| +0 | -0 | -0 | +0 | +0 | -0 |
| NaN | 1 | 1 | 1 | NaN | NaN |

1. **Сравнение чисел**

Недостаток такого подхода в том, что погрешность представления числа увеличивается с ростом самого этого числа. Так, если программа ожидает «10000», то приведенное равенство не будет выполняться для ближайшего соседнего числа (10000,000977). Это особенно актуально, если в программе имеется преобразование из одинарной точности в двойную.

# Погрешности обусловленные форматом с плавающей точкой

Существуют числа, которые нельзя разместить в формате с полями фиксированной длины. Причин этому две. Ограниченная длина поля мантиссы и ограниченная длина поля порядка.

Фиксированная длина поля мантиссы ограничивает точность чисел сохраняемых в формате.

Представление в нормализованной форме гарантирует, что точное число, записанное в формат после округления, имеет точность не меньше точности формата. Представление тех же чисел в ненормализованном виде приводит к снижению их точности до величины меньшей, чем точность формата на величину равную количеству нулей в поле мантиссы слева. Поэтому числа в формате с плавающей точкой следует представлять в нормализованном виде.

# Основные понятия алгебры логики. Способы задания логической функции.

**Основные понятия алгебры логики**

Алгебра логики используется при анализе и синтезе схем ЭВМ по двум причинам. Во*-*первых, это объясняется соответствием представления переменных и функций алгебры логики. Во*-*вторых, двоичным представлением информации и характером работы отдельных компонентов вычислительной техники. Эти компоненты могут пропускать или не пропускать ток, иметь на выходе высокий или низкий уровень сигнала (напряжения или тока).

Приведем основные понятия алгебры логики.

*Логическая переменная —* это такая переменная, которая может принимать одно из двух значений: истинно или ложно (да или нет, единица или ноль).

*Логическая константа —* это такая постоянная величина, значением которой может быть истинно или ложно (да или нет, единица или ноль).

*Логическая функция —* это такая функция, которая может принимать одно из двух значений: истинно или ложно (да или нет, единица или ноль) в зависимости от текущих значений ее аргументов, в качестве которых используются логические переменные.

Способы задания логической функции

Зависимость логической функции от переменных может задаваться по*–*разному:

– словесным описанием;

– таблицей истинности;

– логическим выражением.

*Словесное описание* используется в случае сравнительно несложной логической функции.

*Таблица истинности* является универсальным средством задания логической функции. Она включает все наборы для заданного количества переменных, определяющих значение логической функции, с указанием значений, которые принимает функция для каждого набора. В одной таблице истинности может задаваться несколько логических функций, зависящих от одних и тех же переменных.

*Логическим выражением* называется комбинация логических переменных и констант, связанных элементарными базовыми логическими функциями (или логическими операциями), которые могут разделяться скобками.

Например, логическую функцию *у*1, определенную в вышеприведенной таблице истинности, можно представить в виде логического выражения

 где «+», «·», а также верхнее надчеркивание – знаки базовых логических функций.

Наиболее распространенной в алгебре логики является функционально полная система логических функций, которая в качестве базовых логических функций использует функцию одной переменной «НЕ» (функция отрицания) и две функции двух переменных – «И» (конъюнкция или логическое умножение) и «ИЛИ» (дизъюнкция или логическое сложение). Эта система получила название *система* *булевых* *функций*, или *булевый базис*. В алгебре логики имеется целый раздел *Алгебра Буля*, посвященный этому базису.

# Понятие о принципе двойственности. Суперпозиция логических функций.

**Определение**: Двойственной функцией для f(x1, x1,…,xn) называется функция f\* = ¬f(¬x1, ¬x2,…,¬xn) ¬ - отрицание

**Теорема**: Функция, двойственная к суперпозиции функций, есть суперпозиции функций, двойственных к функциям, составляющим эту суперпозицию.

**Принцип двойственности**

Если функция f задана формулой, построенной с помощью &,∨,¬,0,1 и переменных, то по теореме о суперпозиции двойственных функций и ввиду того, что для функций x&y, ¬x, ,1,0 двойственными являются x∨y ,x ,0,1 соответственно, то f\* получается из f заменой & на ∨, 0 на 1 и т.д. при сохранении исходной расстановки скобок.

# Нормальная и совершенные нормальные логических функций.

Конъюнкция (дизъюнкция), в которой каждая переменная с инверсией или без встречается не более одного раза, называется *элементарной*. Число входящих в нее переменных определяет *ранг* конъюнкции (дизъюнкции).

Дизъюнкция любого числа элементарных конъюнкций называется *дизъюнктивной нормальной формой*(ДНФ), например:

https://helpiks.org/helpiksorg/baza1/703749610099.files/image926.gif .

Конъюнкция любого числа элементарных дизъюнкций называется *конъюнктивной нормальной формой* (КНФ). Например:

https://helpiks.org/helpiksorg/baza1/703749610099.files/image928.gif .

Для каждой функции может существовать несколько равносильных ДНФ (или КНФ), например:

https://helpiks.org/helpiksorg/baza1/703749610099.files/image930.gif .

ДНФ (КНФ), содержащая наименьшее число переменных по сравнению с другими равносильными ДНФ (КНФ), называется *минимальной*.

ДНФ логической функции, состоящая из элементарных конъюнкций одинакового ранга, называется *совершенной дизъюнктивной нормальной формой* (СДНФ). В СДНФ каждая элементарная конъюнкция включает все переменные с инверсиями или без, причем одинаковых конъюнкций нет, например:

https://helpiks.org/helpiksorg/baza1/703749610099.files/image932.gif

От любой ДНФ можно перейти к СДНФ. Для этого необходимо:

- вести недостающие переменные в конъюнкции младших рангов умножением их на равносильность вида формула (х - недостающая переменная);

- раскрыть скобки и избавиться от повторяющихся конъюнкций в соответствии с правилом х+х=х.

СДНФ обладает следующими свойствами:

- если при каком-то наборе f = 1, то СДНФ только одна из элементарных конъюнкций принимает значение единицы;

- если для данного набора f = 0, то в СДНФ ни один из членов не будет равен единице.

Свойства СКНФ:

- если для данного набора f=0, то в СКНФ только одна из элементарных конъюнкций принимает нулевое значение;

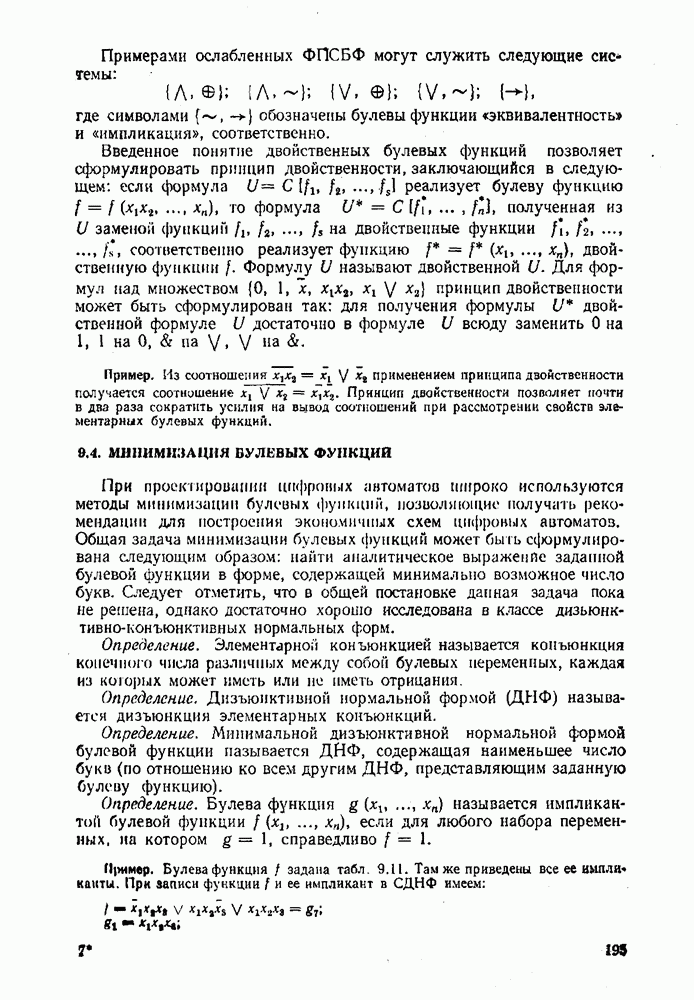
- если для данного набора f=1, то в СКНФ ни один из членов не принимает нулевое значения.

# Минимизация булевых функций. Основные понятия. Наиболее известные методы минимизации. Минимизация системы логических функций. Минимизация частично определенных функций.

**Минимизация булевых операций**

Минимизации позволяют получать рекомендации для построения экономичных схем цифровых автоматов. Общая задача минимизации булевых функций может быть сформулирована следующим образом: найти аналитическое выражение заданной булевой функции в форме, содержащей минимально возможное число букв.

**Основные понятия**



**Наиболее известные методы минимизации**

Имеется несколько методов минимизации ЛФ. Наиболее известны три:

1. расчетный (метод непосредственных преобразований);
2. расчетно-табличный (метод Квайна – Мак-Класки);
3. табличный (метод Вейча–Карно).

**Минимизация системы логических функций**

Минимизация производится:

1. алгебраическими способами;

2. методом карт Карно.

**Минимизация частично определенных функций**

В классе ДНФ

1. Строим СДНФ функции *F*0 .

2. Строим сокращенную ДНФ функции *F*1 .

3. С помощью матрицы покрытий коституент единицы функции *F*0 простыми импликантами функции *F*1 и решеточного выражения строим все тупиковые ДНФ (для некоторых доопределений функции *F* ) .

4. Среди полученных ТДНФ выбираем простейшие, они являются минимальными ДНФ ( для некоторых доопределений функции *F* ) .

В классе КНФ

1. Построение минимальных КНФ для частично определенной функции аналогично построению минимальных КНФ для всюду определенной функции.

2. Алгоритм минимизации частично определенных функций в классе нормальных форм аналогичен алгоритму минимизации в классе нормальных форм для всюду определенных функций.

# Минимизация логических выражений методом Квайна.

Метод Квайна выполняется в два этапа.

*Первый этап* имеет своей целью получение тупиковой формы, представляющей собой дизъюнкцию, в качестве слагаемых которой используются конъюнкции (каждая из них не склеивается ни с одной другой конъюнкцией, входящей в это выражение). Такие конъюнкции называются *простыми импликантами*.

Данный этап выполняется за счет реализации отдельных шагов. На каждом шаге на основании выражения, полученного на предыдущем шаге, выполняются все возможные операции склеивания для пар имеющихся конъюнкций. Каждый шаг понижает ранг исходных конъюнкций на единицу. Шаги повторяются до получения тупиковой формы.

*Второй этап* имеет своей целью устранение из тупиковой формы всех избыточных простых импликант, что дает в результате минимальное логическое выражение.

*Пример:*

Найти методом Квайна минимальное выражение для функции *y*:

**

*Решение*

1*-*й этап:

**

Над конъюнкциями проставлены их номера; в скобках под каждой конъюнкцией (*i–j*) указывают, что данная конъюнкция является результатом склеивания *i-*й и *j-*й конъюнкций исходного выражения.

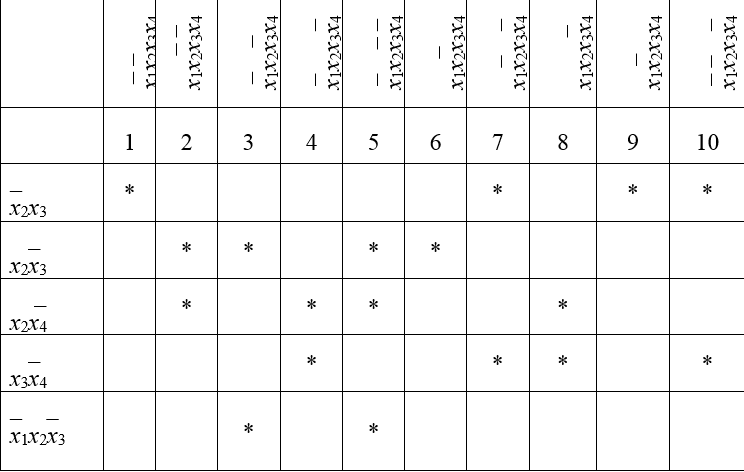
К результатам склеивания логически добавлен ни с чем не склеенный пятый член исходного выражения; несколько одинаковых конъюнкций представляются одной конъюнкцией.

Последнее выражение получено из предыдущего посредством удаления повторяющихся членов.

2*-*й этап:

На основании исходного выражения и полученной тупиковой формы составляется и заполняется импликантная таблица (табл.2.7).

Таблица 2.7

**Импликантная таблица**

Звездочками в каждой строке отмечены те конституенты единицы, которые покрываются соответствующей простой импликантой (практически отмечаются те конституенты единицы, которые включают простую импликанту как свою составную часть).

Анализируя покрытия простыми импликантами конституент единицы заданной функции, составляем ее минимальное выражение:

.

# Минимизация логических выражений с использованием Карт Карно (диаграммами Вейча).

Карта Карно для «*n*» логических переменных представляет собой множество квадратов (клеток), объединенных в близкую к квадрату прямоугольную форму. Каждая такая клетка соответствует одному набору логических переменных, причем наборы двух соседних клеток должны отличаться на значение одной переменной (их наборы образуют склеивающиеся конъюнкции).

Для выполнения минимизации представленной в карте Карно функции необходимо выполнить два этапа:

* охватить множество клеток карты Карно контурами;
* записать минимальное выражение для заданной функции в виде дизъюнкции конъюнкций, где каждая конъюнкция соответствует одному из введенных на карте контуров.

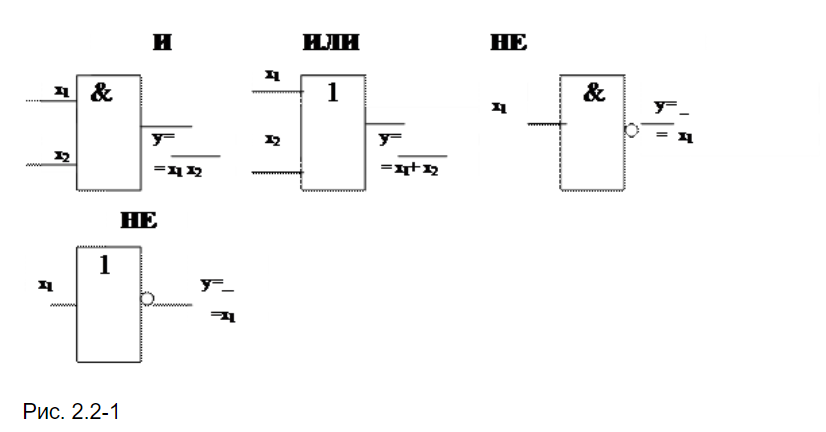
Охват клеток карты контурами выполняется с соблюдением следующих правил:

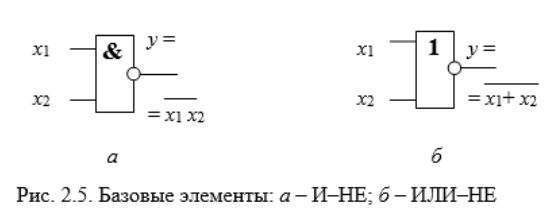
* контур должен иметь прямоугольную форму;
* в контур может входить количество клеток, равное целой степени числа «2»;
* в контур могут входить клетки, являющиеся логическими соседями;
* в контур необходимо включить максимальное количество клеток с учетом вышеприведенных требований;
* контурами необходимо охватить все клетки с единичными значениями;
* контуров должно быть минимальное количество;
* количество клеток в контуре должно быть равно 2ΔR, где ΔR –разность ранга (дельта ранга) конституент единицы заданной функции и ранга конъюнкции, соответствующей контуру.

*Логическими соседями* являются такие две клетки, наборы которых отличаются только одной переменной – в одном эта переменная должна иметь прямое, в другом – обратное значение.

# Синтез логических схем по логическим выражениям в булевом базисе. Логический базис И-НЕ. Логический базис ИЛИ-НЕ.

Логические схемы строятся на основе логических элементов, набор которых определяется заданным логическим базисом. Для базиса Буля в качестве логических элементов используются элементы, реализующие базовые логические функции И, ИЛИ, НЕ.

****

****Булевый базис не является единственной функционально полной системой логических функций. Среди других наибольшее распространение получили базис И–НЕ и базис ИЛИ–НЕ.

При синтезе логических схем в заданном базисе логических элементов (например, в базисах И–НЕ, или ИЛИ–НЕ) целесообразно предварительно исходное выражение привести к форме, в которой в выражении будут использованы только логические операции, соответствующие используемым логическим элементам в заданном базисе.

# Законы и правила алгебры Буля.

В алгебре Буля логические выражения включают логические операции И, ИЛИ, НЕ, которые могут быть использованы в самых различных сочетаниях. При оценке значения такого выражения необходимо решить его для конкретного набора переменных. В алгебре Буля применяется следующая приоритетность выполнения операций: сначала рассчитываются значения имеющих место отрицаний и скобок, затем выполняется операция И (логическое умножение); самый низший приоритет имеет операция ИЛИ (логическая сумма).

При работе с булевыми логическим выражениями используются следующие законы, правила и операции.

**Переместительный (коммутативный) закон.**

– от перемены мест логических слагаемых сумма не меняется

х1 + х2 + х3 + х4 = х4 + х3 + х2 + х1

– от перемены мест логических сомножителей их произведение не меняется

х1х2х3х4 = х4х3х2х1

1. Сочетательный (ассоциативный) закон.

– при логическом сложении отдельные слагаемые можно заменить их суммой

х1 + х2 + х3 + х4 = (х2 + х3) + х1 + х4 = (х1 + х4 ) + (х2 + х3)

– при логическом умножении отдельные логические сомножители можно заменить их произведением

х1 х2 х3 х4 = (х2 х3)х1х4 = (х1 х4) (х2 х3)

1. Распределительный (дистрибутивный) закон.

(х1 + х2) х3 = х1х3 + х2х3;

(х1 + х2) (х1 + х3) = х1 + х2х3

1. Правило де Моргана.

– отрицание суммы равно произведению отрицаний

– отрицание произведения равно сумме отрицаний

1. Операция склеивания.

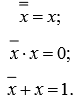
– операция склеивания для конъюнкций, где А – переменная или любое логическое выражение

– операция склеивания для дизъюнкций

Если в качестве А используется простая конъюнкция, т. е. конъюнкция, представляющая собой логическое произведение переменных и их отрицаний, то имеет место



5) Операции с отрицаниями.

– двойное отрицание равносильно отсутствию отрицания

Операции с константами.



Операции с одинаковыми операндами.





# Параллелизм

Параллелизм – основа высокопроизводительной работы всех подсистем вычислительных машин. Организация памяти любого уровня иерархии, организация системного ввода/вывода, организация мультиплексирования шин и т.д. базируются на принципах параллельной обработки запросов. Современные операционные системы являются многозадачными и многопользовательскими, имитируя параллельное исполнение программ посредством механизма прерываний.

Уровни параллелизма

1. Микроуровневый (команды разд. на фазы)
2. Уровень команд (параллельное выполнение нескольких команд)
3. Уровень потоков (задачи разб. на части, которые выполн. парал.)
4. Уровень заданий (независ. задания одноврем. на неск. процессор.)

# Устройства ЭВМ. Состав АЛУ.

**Устройства ЭВМ**

Классическая ЭВМ состоит из трех основных устройств: арифметико-логического устройства, устройства управления и запоминающего устройства.

**Состав АЛУ**

АЛУ состоит из следующих типовых узлов:

– Регистры (R), служащие для хранения операндов и результатов;

– Сумматор (SM), служащий для выполнения операции суммирования многоразрядных кодов;

– Операционные узлы (ОУ), служащие для выполнения логических операций;

– Мультиплексор (MS);

– Счетчик (Сч), обеспечивающий подсчет тактов длинных операций;

– Регистр флажков (RF), служащий для фиксации особой информации, характеризующей полученный результат.

# Типы памяти.

Памятью компьютера называется совокупность устройств, предназначенная для хранения программ, вводимой информации, промежуточных результатов и выходных данных.

|  |
| --- |
| Память  Внутренняя память |

|  |
| --- |
| Внешняя память |

|  |
| --- |
| Энергонезависимая |

|  |
| --- |
| Энергозависимая |

|  |
| --- |
| ПЗУ |

|  |
| --- |
| ОЗУ |

|  |
| --- |
| Кэш-память |

|  |
| --- |
| Видеопамять |

|  |
| --- |
| Память с последовательным доступом |

|  |
| --- |
| Память с произвольным доступом |

|  |
| --- |
| ПЛ |

|  |
| --- |
| ПК |

|  |
| --- |
| НМЛ |

|  |
| --- |
| НГМД |

|  |
| --- |
| НЖМД |

|  |
| --- |
| CD-ROM |

# Код Грея.

Код Гре́я — двоичный код, в котором две «соседние» (в упорядоченном, то есть лексикографическом, наборе) кодовые комбинации различаются только цифрой в одном двоичном разряде.

Двоичным кодом Грея порядка n называется последовательность всех image n-битных кодов, в которой любые два соседних кода различаются ровно в одном разряде.  
  
Пример кодов Грея порядка 2:

* 00
* 01
* 11
* 10

Наиболее часто на практике применяется рефлексивный двоичный код Грея, хотя в общем случае существует бесконечное множество кодов Грея со значениями цифр в разрядах, взятых из различных алфавитов. В большинстве случаев, под термином «код Грея» понимают именно рефлексивный бинарный код Грея.

Преобразование двоичного кода в код Грея

Коды Грея легко получаются из двоичных чисел путём побитовой операции «Исключающее ИЛИ» с тем же числом, сдвинутым вправо на один бит и в котором старший разряд заполняется нулём. Следовательно, i-й бит кода Грея Gi выражается через биты двоичного кода Bi следующим образом:



где  — операция «исключающее ИЛИ»; биты нумеруются справа налево, начиная с младшего.

Преобразование кода Грея в двоичный код

Обратный алгоритм — преобразование кода Грея в двоичный код — можно выразить рекуррентной формулой

Действительно, если подставить в эту формулу вышеприведённое выражение для i-го бита кода Грея, получим



# Обратная польская запись.

Обра́тная по́льская за́пись (англ. Reverse Polish notation, RPN) — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций.

Особенности обратной польской записи следующие:

* Порядок выполнения операций однозначно задаётся порядком следования знаков операций в выражении, поэтому отпадает необходимость использования скобок и введения приоритетов и ассоциативности операций.
* В отличие от инфиксной записи, невозможно использовать одни и те же знаки для записи унарных и бинарных операций. Так, в инфиксной записи выражение 5 \* (−3 + 8) использует знак «минус» как символ унарной операции (изменение знака числа), а выражение (10 − 15) \* 3 применяет этот же знак для обозначения бинарной операции (вычитание). Конкретная операция определяется тем, в какой позиции находится знак. Обратная польская запись не позволяет этого: запись 5 3 − 8 + \* (условный аналог первого выражения) будет интерпретирована как ошибочная, поскольку невозможно определить, что «минус» после 5 и 3 обозначает не вычитание; в результате будет сделана попытка вычислить сначала 5 − 3, затем 2 + 8, после чего выяснится, что для операции умножения не хватает операндов. Чтобы всё же записать это выражение, придётся либо переформулировать его (например, записав вместо выражения − 3 выражение 0 − 3), либо ввести для операции изменения знака отдельное обозначение, например, «±»: 5 3 ± 8 + \*.
* Так же, как и в инфиксной нотации, в ОПН одно и то же вычисление может быть записано в нескольких разных вариантах. Например, выражение (10 − 15) \* 3 в ОПН можно записать как 10 15 − 3 \*, а можно — как 3 10 15 − \*
* Из-за отсутствия скобок обратная польская запись короче инфиксной. За этот счёт при вычислениях на калькуляторах повышается скорость работы оператора (уменьшается количество нажимаемых клавиш), а в программируемых устройствах сокращается объём тех частей программы, которые описывают вычисления. Последнее может быть немаловажно для портативных и встроенных вычислительных устройств, имеющих жёсткие ограничения на объём памяти.

Общий порядок

Автоматизация вычисления выражений в обратной польской нотации основана на использовании [стека](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA). Алгоритм вычисления для стековой машины элементарен:

1. Обработка входного символа
   * Если на вход подан операнд, он помещается на вершину стека.
   * Если на вход подан знак операции, то соответствующая операция выполняется над требуемым количеством значений, извлечённых из стека, взятых в порядке добавления. Результат выполненной операции кладётся на вершину стека.
2. Если входной набор символов обработан не полностью, перейти к шагу 1.
3. После полной обработки входного набора символов результат вычисления выражения лежит на вершине стека.

# АЦП и ЦАП.

ЦАП – цифро-аналоговые преобразователи – устройства, предназначенные для преобразования дискретного (цифрового) сигнала в непрерывный (аналоговый) сигнал. Преобразование производится пропорционально двоичному коду сигнала.

АЦП – аналого-цифровые преобразователи – устройства, предназначенные для преобразования непрерывных (аналоговых) сигналов в цифровые.

ЦАП (цифро-аналоговый преобразователь) и АЦП (аналого-цифровой преобразователь) входят в состав кодека и непосредственно выполняют дискретизацию:

— при воспроизведении ЦАП переводит цифровой код в аналоговый сигнал,  
— при записи АЦП выполняет обратное преобразование.

Чем качественнее АЦП, тем более чистый и детальный звук будет литься из колонок.

Чем качественнее ЦАП, тем точнее будет переведен в цифру аналоговый сигнал.

АЦП и ЦАП являются основными устройствами ввода-вывода ин­формации в цифровых системах, предназначенных для обработки анало­говой информации или управления каким-либо технологическим процес­сом.

*Важнейшие характеристики АЦП и ЦАП:*

1) Вид аналоговой величины, являющейся входной для АЦП и выходной для ЦАП (напряжение, ток, временной интервал, фаза, частота, угловое и линейное перемещение, освещенность, давление, темпе­ратура и т.п.).

2) Разрешающая способность и точность преобразования (разре­шающая способность определяется количеством двоичных разрядов кода или возможным количеством уровней аналогового сигна­ла, точность определяется наибольшим значением отклонения аналогового сигнала от цифрового и наоборот).

3) Быстродействие, определяемое интервалом времени от момента подачи сигнала опроса (запуска) до момента достижения выход­ным сигналом установившегося значения (ед. микросекунд, десят­ки наносекунд)

В любом преобразователе выделяют цифровую и аналоговую части. В цифровой производятся кодирование и декодирование цифровых сигна­лов, их запоминание, счет, цифровое компарирование (сравнение), выра­ботка логических сигналов управления. Для этого используют: дешифра­торы, мультиплексоры, регистры, счетчики, цифровые компараторы, логические элементы.

В аналоговой части преобразователя производятся операции: усиле­ния, сравнения, коммутации, сложения и вычитания аналоговых сигна­лов. Для этого используются аналоговые элементы: ОУ, аналоговые ком­параторы, ключи и коммутаторы, резистивные матрицы и т.д.

38. Корректирующие коды. Код Хэ́мминга. Область применения.

**Корректирующие коды**

Корректирующий код (также помехоустойчивый код) — [код](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4), предназначенный для [обнаружения и исправления ошибок](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8C_%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA).

Фактически любой код, исправляющий ошибки, может быть также использован для обнаружения ошибок (при этом он будет способен обнаружить бо́льшее число ошибок, чем был способен исправить). Коды, исправляющие ошибки, применяются в системах [цифровой связи](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C), в том числе: спутниковой, радиорелейной, сотовой, передаче данных по телефонным каналам, а также в системах хранения информации, в том числе магнитных и оптических. Коды, обнаруживающие ошибки, применяются в [сетевых протоколах](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) различных [уровней](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_OSI).

**Код Хэмминга**

Код Хэмминга — вероятно, наиболее известный из первых самоконтролирующихся и самокорректирующихся кодов. Построены они применительно к двоичной системе счисления.

**Применение**

Код Хэмминга используется в некоторых прикладных программах в области хранения данных, особенно в RAID 2; кроме того, метод Хэмминга давно применяется в памяти типа ECC и позволяет «на лету» исправлять однократные и обнаруживать двукратные ошибки.

# Языки описания аппаратуры. ПЛИС (FPGA) модули.

**Языки описания аппаратуры**

Язык описания аппаратуры (HDL от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) hardware description language) — специализированный компьютерный язык, используемый для описания структуры и поведения [электронных схем](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%85%D0%B5%D0%BC%D0%B0), чаще всего [цифровых логических](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B2%D0%B5%D0%BD%D1%82%D0%B8%D0%BB%D1%8C) схем.

Языки описания аппаратуры внешне похожи на такие [языки программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), как [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) или [Паскаль](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D1%8C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), написанные на них программы также состоят из выражений, операторов, управляющих структур.

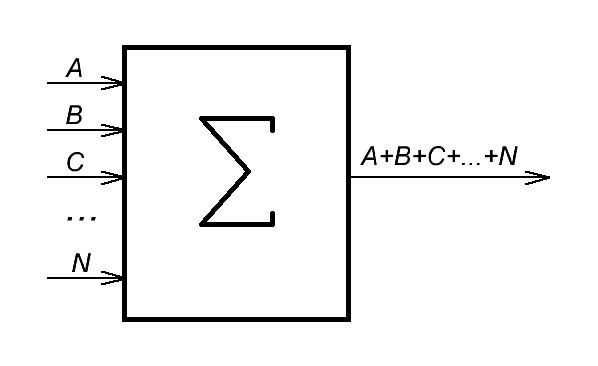
**ПЛИС (FPGA) модули**

ПЛИС (ПЛИС, англ. programmable logic device, PLD) — электронный компонент, используемый для создания цифровых интегральных схем. В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задаётся посредством программирования.

Структура ПЛИС основана на ФАЛ (булевой алгебры) и содержит следующие логические элементы: — логическое «И»(коньюнкторы); — логическое «ИЛИ» (дизъюнкторы); — логическое «НЕ» (инверторы); — буферные элементы с прямыми, инверсными и тристабильными выходами; — «Исключающее ИЛИ»; — триггеры D- и T-типа; — мультиплексоры конфигурации.

# Сумматор. Многоразрядный сумматор.

**Сумматор**

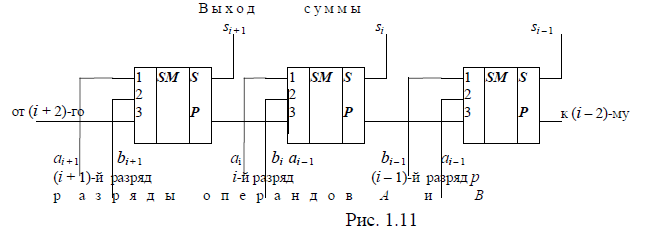
Сумма́тор — устройство, преобразующее [информационные сигналы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB) (аналоговые или цифровые) в сигнал, эквивалентный сумме этих сигналов; устройство, производящее операцию [сложения](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5). Т.е. сумматорами называют логические устройства, выполняющие арифметические суммирование кодов двоичного числа.

В электронике сумматоры делятся на две группы:

* сумматоры аналоговых сигналов
* сумматоры цифровых сигналов

**Многоразрядный сумматор**

Многоразрядный двоичный сумматор строится на основе одноразрядных сумматоров с введением соответствующих связей между разрядами.



Приведенная схема многоразрядного сумматора называется схемой сумматора с последовательным переносом. Схема очень простая. Сумматор обладает малым быстродействием из-за последовательного учета переноса, возникшего в младшем разряде, в непрерывной цепочке старших разрядов, имеющих значение поразрядной суммы, равное единице. Такие разряды называются разрядами, пропускающими перенос. В худшем случае перенос, возникший в младшем разряде, распространяется до самого старшего разряда формируемой суммы.

# Полная система логических функций.

Функционально полная система логических элементов — это такой набор элементов, используя который можно реализовать любую сколь угодно сложную логическую функцию. Поскольку любая логическая функция представляет собой комбинацию простейших функций — дизъюнкции, конъюнкции и инверсии, то набор из элементов трех типов, реализующих соответственно функции И, или и НЕ, естественно, является функционально полным.

Система функций называется полной, если любую логическую функцию можно представить в виде формулы, содержащей функции этой системы. Имеет место теорема Поста: Для того чтобы некоторый набор функций был полным, необходимо и достаточно, чтобы в него входили функции, не принадлежащие каждому из классов Т0, Т1, L, M, S. Т0 – класс функций сохраняющих 0. В него входят функции, у которых f (0,0,...,0) = 0 . Т1 – класс функций, сохраняющих 1, т.е. таких у которых f (1,1,...,1) = 1.

# Искусство управления сложностью. Цифровая абстракция.

**Искусство управления сложностью**

Одной из характеристик, отличающих профессионального инженера-электронщика или программиста от дилетанта, является систематический подход к управлению сложностью многоуровневой системы.

Критически важный принцип управления сложностью системы – абстракция, подразумевающая исключение из рассмотрения тех элементов, которые в данном конкретном случае несущественны для понимания работы этой системы. Любую систему можно рассматривать с различных уровней абстракции.

****

# Логические элементы. Таблицы истинности. Обозначения элементов в разных представления.

**Логические элементы**

Элементы ЭВМ, являющиеся представителями низшего иерархического уровня архитектуры компьютера – цифрового логического уровня (схемотехнического), реализованные на радиотехнических деталях, представляют собой мельчайшие компоненты, на основе которых строятся более крупные составляющие вычислительной машины.

Можно выделить три основные разновидности элементов – логические элементы, запоминающие, специальные.

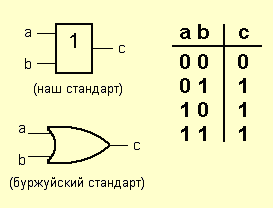
**Логический элемент** — элемент, осуществляющий определенные логические зависимость между входными и выходными сигналами.

**Логический элемент** – это такая схемка, у которой несколько входов и один выход. Каждому состоянию сигналов на входах, соответствует определенный сигнал на выходе.

**Таблицы истинности**

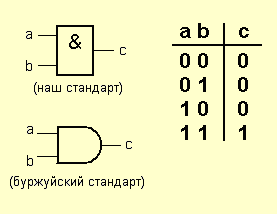
Для того, чтобы понять как он работает, нужно нарисовать таблицу, в которой будут перечислены состояния на выходе при любой комбинации входных сигналов. Такая таблица называется «**таблица истинности**». Таблицы истинности широко применяются в цифровой технике для описания работы логических схем.

**Обозначения элементов в разных представлении**

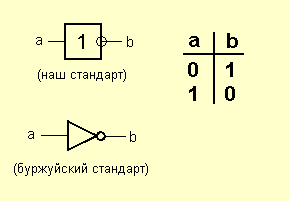
**Элемент «И» (AND)**

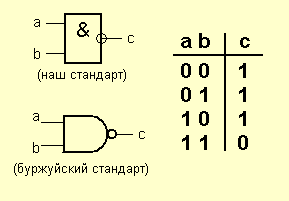
**Элемент «ИЛИ» (OR)**

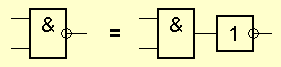
По другому, его зовут «дизъюнктор».



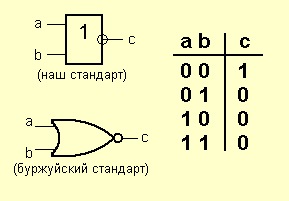
**Элемент «НЕ» (NOT)**

Чаще, его называют «инвертор».

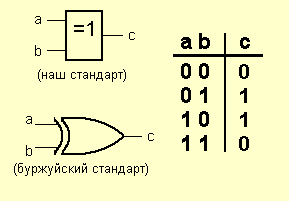
**Элемент «И-НЕ» (NAND)**



**Элемент «ИЛИ-НЕ» (NOR)**



**Элемент «Исключающее ИЛИ» (XOR)**



# За пределами цифровой абстракции. Напряжение питания. Логические уровни. Допускаемые уровни шумов.

**За пределами цифровой абстракции**

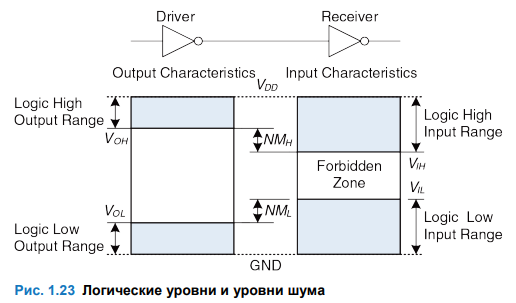
Цифровая система оперирует дискретными переменными. Однако для представления этих переменных используются непрерывные физические величины, такие как напряжение в электрической цепи, положение шестеренок в механической передаче или уровень жидкости в гидравлическом цилиндре. Задача разработчика цифровой системы – определить, каким образом непрерывно-меняющаяся величина соотносится с конкретным значением дискретной переменной. Рассмотрим, например, задачу представления двоичного сигнала А напряжением в электрической цепи. Допустим, что напряжение 0 В соответствует значению А = 0, а напряжение 5 В соответствует А = 1. Однако, реальная цифровая система должна быть устойчива к неизбежному в такой ситуации шуму, так что значение 4,97 В, вероятно, также следует толковать как A = 1. А что делать, если напряжение равно 4,3 В? Или 2,8 В? Или 2,500000 В?

**Напряжение питания**

Предположим, что минимальное напряжение в электронной цифровой системе, называемое также напряжением земли (ground voltage, или просто ground, или GND), составляет 0 В. Самое высокое напряжение в системе поступает от блока питания и, как правило, обозначается VDD. Транзисторные технологии семидесятых и восьмидесятых годов прошлого века в основном использовали VDD равное 5 В. С переходом на транзисторы меньшего размера, VDD последовательно снижали до 3,3 В, 2,5 В, 1,8 В, 1,5 В, 1,2 В и даже ниже для экономии электроэнергии и во избежание перегрузки транзисторов.

**Логические уровни**

Первый логический элемент в рассматриваемой схеме называется источник (driver), а второй – приемник (receiver). Выходной сигнал источника подключается ко входу приемника. Источник выдает выходной сигнал низкого напряжения (0) в диапазоне от 0 В до VOL или выходной сигнал высокого напряжения (1) в диапазоне от VOH до VDD. Если приемник получает на вход сигнал в диапазоне от 0 до VIL, он рассматривает такой сигнал как нуль. Если приемник получает на вход сигнал в диапазоне от VIH до VDD, он рассматривает такой сигнал как единицу. Если же по какой-либо причине, например, наличия шумов или неисправности одного из элементов схемы, напряжение сигнала на входе приемника падает настолько, что попадает в запретную зону (forbidden zone) между VIL и VIH, то поведение этого логического элемента становится непредсказуемым. VOH и VOL называются соответственно высоким и низким логическими уровнями выхода (output high and low logic levels), а VIH и VIL называются соответственно высоким и низким логическими уровнями входа (input high and low logic levels).

****

**Допускаемые Уровни Шумов**

Для того чтобы выходной сигнал источника был правильно интерпретирован на входе приемника, необходимо, чтобы VO < VIL и VOH > VIH. В этом случае, даже если выходной сигнал источника будет загрязнен шумами, приемник по-прежнему сможет правильно определить логический уровень входного сигнала. Допускаемый уровень шумов (noise margin) – это то максимальное количество шума, присутствие которого в выходном сигнале источника не мешает приемнику корректно интерпретировать значение полученного сигнала. Как можно увидеть на Рис. 1.23, значения нижнего допускаемого уровня шумов (low noise margin) и верхнего допускаемого уровня шумов (high noise margin) определяются следующим образом:

NML = VIL – VOL (1.2)

NMH = VO – VIH (1.3)

# Передаточная характеристика. Статическая дисциплина.

**Передаточная Характеристика**

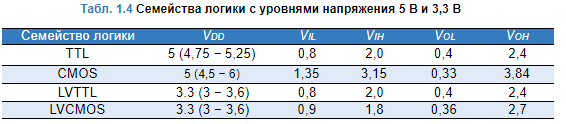
Для понимания предела цифровой абстракции мы должны рассмотреть поведение логических вентилей с аналоговой точки зрения.

Передаточная характеристика (DС transfer characteristics) какого-либо логического вентиля описывает напряжение на выходе этого элемента как функцию напряжения на его входе, когда входной сигнал изменяется настолько медленно, что выходной сигнал успевает изменяться вслед за ним. Такая характеристика называется передаточной, поскольку описывает взаимосвязь между входным и выходным напряжением.

**Статическая Дисциплина**

Для того, чтобы избежать попадания входных сигналов в запретные зоны, логические вентили должны разрабатываться в соответствии с принципом статической дисциплины (static discipline). Принцип статической дисциплины требует, чтобы при условии наличия логически корректных сигналов на входе каждый элемент системы выдавал логически корректные сигналы на выходе.

Применение принципа статической дисциплины ограничивает свободу разработчика в выборе аналоговых элементов для построения цифровых систем, однако помогает обеспечить простоту и надежность разрабатываемых цифровых схем. Используя этот принцип, разработчик поднимается с аналогового уровня абстракции на цифровой, что увеличивает производительность проектировщика, избавляя его от рассмотрения излишних деталей.

****

# Биполярные и КМОП транзисторы. Полупроводники. Конденсаторы. n-МОП и p-МОП-транзисторы.

**Биполярные и КМОП транзисторы**

Биполярный транзистор – это полупроводниковый прибор с двумя взаимодействующими между собой р-п–переходами и тремя или более выводами.

КМОП (К-МОП; комплементарная логика на [транзисторах](https://science.fandom.com/ru/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B8%D1%81%D1%82%D0%BE%D1%80) металл-оксид-полупроводник; [англ.](https://science.fandom.com/ru/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) CMOS, Complementary-symmetry/metal-oxide semiconductor) — технология построения электронных схем.

**Полупроводники**

Полупроводни́к — [материал](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D1%80%D0%B8%D0%B0%D0%BB), по [удельной проводимости](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B2%D0%BE%D0%B4%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D1%8C) занимающий промежуточное место между [проводниками](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B2%D0%BE%D0%B4%D0%BD%D0%B8%D0%BA_(%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%BE)) и [диэлектриками](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D0%BA), и отличающийся от проводников сильной зависимостью [удельной проводимости](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B2%D0%BE%D0%B4%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D1%8C) от концентрации примесей, температуры и воздействия различных видов [излучения](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D0%BB%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Основным свойством полупроводников является увеличение электрической проводимости с ростом температуры.

**Конденсаторы**

**Конденсатор** - представляет из себя пассивный радиоэлемент, который обладает таким свойством, как сохранение электрического заряда на своих обкладках, если, конечно, перед этим его зарядить каким-нибудь источником питания.

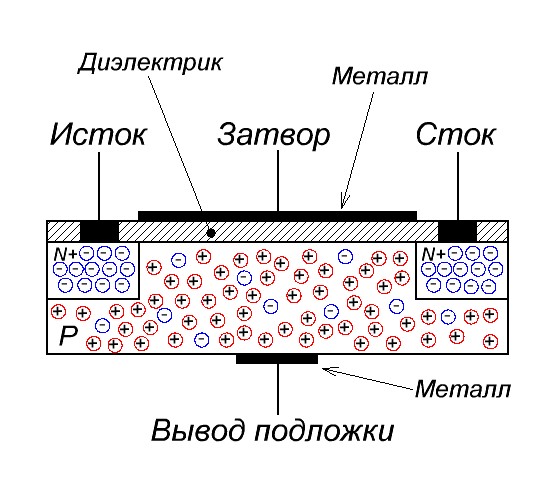
Грубо говоря, конденсатор можно рассматривать как батарейку или аккумулятор электрической энергии. Но вся разница в том, что аккумулятор или батарейка имеют в своем составе источник [ЭДС](https://www.ruselectronic.com/eds-elektrodvizhushchaya-sila/), тогда как конденсатор лишен этого внутреннего источника.

**n-МОП и p-МОП-транзисторы.**

Как часто вы слышали название полевой транзистор МОП, MOSFET, MOS, полевик, МДП-транзистор, транзистор с изолированным затвором? Это все слова синонимы и относятся к одному и тому же радиоэлементу: полевому МОП-транзистору.

Полное название такого радиоэлемента на английский манер звучит как Metal Oxide Semiconductor Field Effect Transistors (MOSFET), что в дословном переводе Металл Оксид Полупроводник Поле Влияние Транзистор. Если преобразовать на наш могучий русский язык, то получается как полевой транзистор со структурой Металл Оксид Полупроводник или просто МОП-транзистор. Почему МОП-транзистор также называют МДП-транзистором и транзистором с изолированным затвором.

Если «разрезать» МОП-транзистор, то можно увидеть вот такую картину:



# Логический вентиль НЕ и другие на КМОП-транзисторах. Псевдо n-МОП-Логика. Потребляемая Мощность.

**Логический вентиль НЕ и другие на КМОП-транзисторах**

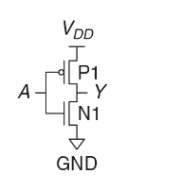
Логи́ческий ве́нтиль — базовый элемент цифровой схемы, выполняющий элементарную логическую операцию, преобразуя таким образом множество входных логических сигналов в выходной логический сигнал. Логика работы вентиля основана на [битовых операциях](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D1%8B%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8) с входными цифровыми сигналами в качестве [операндов](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D0%BD%D0%B4).

Схема демонстрирует, как можно построить логический элемент НЕ, используя КМОП-транзисторы. На этой схеме треугольник обозначает напряжение земли GND, а горизонтальная линия обозначает напряжение питания VDD.

**Псевдо n-МОП-Логика**

При использовании псевдо n-МОП-логики (pseudo-nMOS logic), или просто псевдо-логики, медленный стек из p-МОП транзисторов заменяют одним «слабым» p-МОП транзистором, который всегда находится во включенном состоянии. Такой транзистор часто называют слабым подтягивающим транзистором (weak pull-up). Физические параметры p-МОП-транзистора подбираются таким образом, что этот транзистор до высокого логического уровня (1) выход Y «подтягивает слабо» – то есть только в том случае, когда все n-МОП-транзисторы выключены. Но если при этом хотя бы один из n-МОП-транзисторов включается, то он, превосходя по мощности слабый подтягивающий транзистор, «перетягивает» выход Y настолько близко к напряжению земли GND, что на выходе получается логический 0.

**Потребляемая мощность**

Потребляемая мощность – это количество энергии, потребляемой системой в единицу времени. Именно потребляемая мощность определяет время автономной работы без подзарядки батареи любого портативного устройства, такого как сотовый телефон или ноутбук. Не стоит думать, однако, что потребляемая мощность – второстепенный параметр для стационарных устройств.

Цифровая система потребляет энергию как в динамическом режиме, когда выполняет какие-либо операции, так и в статическом, когда система находится в состоянии покоя (idle). В динамическом режиме энергия расходуется на зарядку емкостей элементов системы, когда эти элементы переключаются между 0 и 1.

# Проектирование комбинационной логики. От логики к логическим элементам. Что такое Х и Z: способы сопряжения микросхем в ЭВМ.

**Проектирование комбинационной логики**

Правила комбинационной композиции говорят нам, как мы можем построить большую комбинационную схему из более маленьких комбинационных элементов. Схема является комбинационной, если она состоит из соединенных между собой элементов и выполнены следующие условия:

Каждый элемент схемы сам является комбинационным;

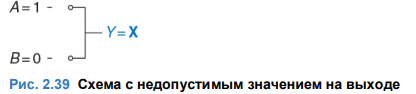
Каждое соединение схемы является или входом, или подсоединено к одному-единственному выходу другого элемента схемы;

Схема не содержит циклических путей: каждый путь в схеме проходит через любое соединение не более одного раза.

**Что за X и Z**

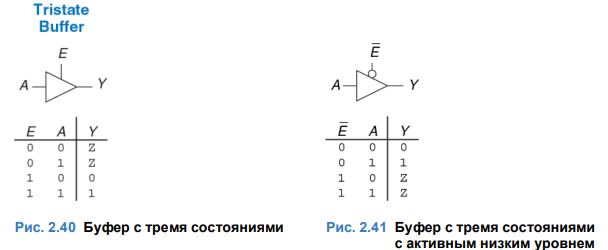
Булева алгебра ограничена значениями 0 и 1. Однако реальные схемы могут также иметь недопустимое и плавающее состояния, представляемые символами X и Z соответственно.

Символ X обозначает неизвестное логическое значение или недопустимое значение физического напряжения в соединении, не соответствующее уровням логических 0 и 1. Это обычно происходит, если к соединению подключены выходы других элементов схемы, выдающие значения 0 и 1 одновременно.



Эта ситуация, называемая состязанием или конфликтом (contention), считается ошибкой, и её необходимо избегать.

Символ Z указывает, что напряжение в цепи не определяется ни источником ВЫСОКОГО, ни источником НИЗКОГО напряжения. Говорят, что такая цепь отключена, находится в состоянии высокого импеданса или в третьем состоянии. Буфер с тремя состояниями имеет вход A, выход Y и сигнал управления E.



# Временные характеристики цифровых микросхем. Задержка распространения и задержка реакции. Импульсные помехи.

**Временные характеристики цифровых микросхем**

Переход от НИЗКОГО уровня к ВЫСОКОМУ называется положительным перепадом или фронтом. Аналогично, переход от ВЫСОКОГО уровня к НИЗКОМУ называется соответственно отрицательным перепадом или срезом. Величина задержки измеряется от момента времени, когда входной сигнал А достигает уровня 50%, до момента достижения уровня 50% выходным сигналом Y. Уровень 50% – это точка, в которой сигнал находится ровно посередине между НИЗКИМ и ВЫСОКИМ логическими уровнями.

**Задержка распространения и задержка реакции**

Комбинационная логика характеризуется задержкой распространения (propagation delay) и задержкой реакции, или отклика (contamination delay). Задержка распространения tpd – это максимальное время от начала изменения входа до момента, когда все выходы достигнут установившихся значений. Задержка реакции tcd – это минимальное время от момента, когда вход изменился, до момента, когда любой из выходов начнет изменять свое значение.

**Импульсные помехи**

Однако может оказаться, что одиночное изменение на входе вызывает несколько выходных изменений. Это называется импульсной помехой или паразитным импульсом. Хотя паразитный импульс обычно не вызывает проблем, важно понимать, что он есть, и уметь распознавать его на временных диаграммах.

# Базовые комбинационные блоки. Мультиплексоры. Логика на мультиплексорах. Дешифраторы.

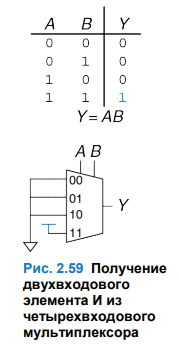
**Базовые комбинационные блоки**

Комбинационные логические элементы часто группируются в «строительные блоки», используемые для создания сложных систем. Это позволяет абстрагироваться от излишней детализации уровня логических элементов и подчеркнуть функцию «строительного блока».

**Мультиплексоры**

**Мультиплексоры** являются одними из наиболее часто используемых комбинационных схем. Они позволяют выбрать одно выходное значение из нескольких входных в зависимости от значения сигнала выбора.

**Логика на мультиплексорах**

****Логика на мультиплексорах Мультиплексоры могут использоваться как таблицы преобразования (lookup tables) для выполнения логических функций. На Рис. 2.59 показан четырехвходовой мультиплексор, используемый для реализации двухвходового элемента И. Входы A и B служат управляющими линиями. Входы данных мультиплексора подключены к 0 и 1 согласно соответствующей строке таблицы истинности. Способ заключается в том, чтобы подать один из литералов, так же как 0 и 1, на вход данных мультиплексора.

Для иллюстрации этого принципа на Рис. 2.60 показаны функции двухвходовых элементов И и ИСКЛЮЧАЮЩЕЕ ИЛИ, реализованных на двухвходовых мультиплексорах.

****

**Дешифраторы**

В общем случае у дешифратора имеется N входов и 2N выходов. Он выдает единицу строго на один из выходов в зависимости от набора входных значений. Выходы образуют прямой унитарный код (one-hot code), называемый так потому, что в любое время только один из выходов может принимать высокий уровень.

# Проектирование последовательностной логики. Защёлки и триггеры. RS-триггер. D-защёлка. D-Триггер. Регистр.

**Проектирование последовательностной логики**

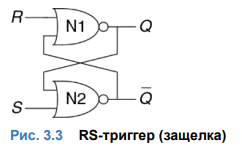
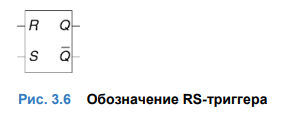
Значение на выходе последовательностной логической схемы зависит как от текущих, так и от предыдущих входных значений, следовательно, последовательнстные логические схемы обладают памятью. Последовательностные логические схемы могут явно запоминать предыдущие значения определенных входов, а могут «сжимать» предыдущие значения определенных входов в меньшее количество информации, называемое состоянием системы. Состояние цифровой последовательностной схемы – набор бит, называемый переменными состояния. Защелки и триггеры - являются простыми последовательностными схемами, запоминающими один бит информации.

**Защёлки и триггеры**

Основным блоком для построения памяти является бистабильная ячейка – элемент с двумя устойчивыми состояниями.

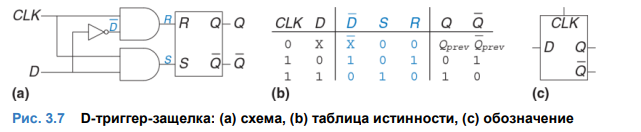
**RS-триггер**

Одной из простейших последовательностных схем является RS-триггер, (от англ. Reset и Set), состоящий из двух перекрестно включенных элементов ИЛИ-НЕ. У защелки есть два входа – R и S и два выхода Q и Q¯ . Принципы работы RS-триггера и схемы с перекрестно включенными инверторами аналогичны, но состояние защелки контролируются R и S входами, которые сбрасывают и устанавливают выход Q.



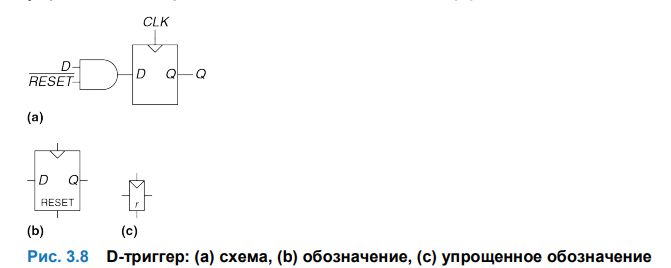
**D-защёлка**

RS-триггер неудобен из-за необычного поведения, если на оба входа триггера одновременно поступает высокий уровень сигнала. Более серьезная проблема состоит в том, что вопросы ЧТО и КОГДА в контексте изменения состояния триггера объединены его R и S входами. Подача логической единицы на эти входы определяет не только, ЧТО произойдет, но и КОГДА это произойдет. Разработка схем упрощается, если эти вопросы ЧТО и КОГДА разделены. D-триггер-защелка решает эти проблемы. У триггера есть два входа: вход данных D, определяющий, каким будет следующее состояние, и вход тактового сигнала CLK, определяющий, когда оно изменится.



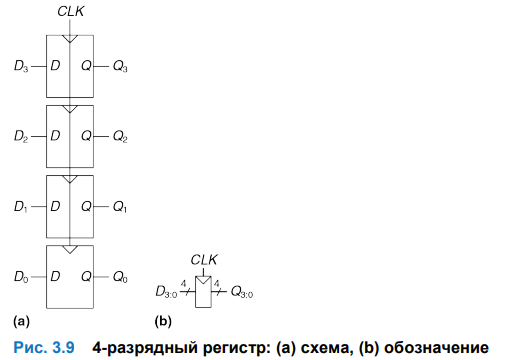
**D-триггер**

D-Триггер может быть построен из двух включенных последовательно D-защелок.

****

**Регистр**

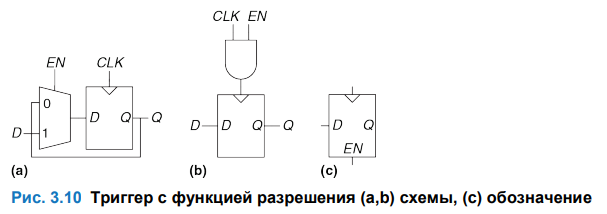
N-разрядный регистр – набор из N триггеров с общим тактовым сигналом. Таким образом, все биты регистра обновляются одновременно. Регистр является ключевым блоком при построении большинства последовательностных схем. На Рис. 3.9 показана схема и обозначение 4-разрядного регистра со входами D3:0 и выходами Q3:0. D3:0 и Q3:0 являются 4-разрядными шинами.

****

# Триггер с функцией разрешения. Триггер с функцией сброса. Проектирование синхронных логических схем. Синхронные последовательностные схемы. Синхронные и асинхронные схемы.

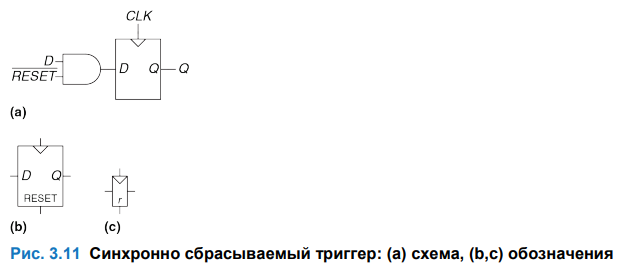
**Триггер с функцией разрешения**

У некоторых триггеров имеется еще один вход, называемый EN, или ENABLE (разрешить). Этот вход определяет, будут ли данные загружены по фронту или нет. Такие триггеры полезны, если мы хотим загружать значения в триггер только на протяжении какого-то времени, а не по каждому фронту тактовому импульсу.

****

**Триггер с функцией сброса**

В триггере с функцией сброса добавляется еще один вход, называемый RESET (сброс). Когда на RESET подан 0, сбрасываемый триггер ведет себя как обычный D-триггер. Когда на RESET подана 1, такой триггер игнорирует вход D и сбрасывает выход в 0. Триггеры с функцией сброса полезны, когда мы хотим ускорить установление определенного состояния (т.е. 0) во всех триггерах системы при первом включении.

****

**Проектирование синхронных логических схем**

Вообще говоря, последовательностные схемы включают в себя все схемы, которые не являются комбинационными, то есть последовательностные схемы – это те, значение выхода которых нельзя однозначно определить, зная лишь текущие значения входов. Поведение некоторых последовательностных схем может быть весьма сложным.

**Синхронные последовательностные схемы**

Это скорее последовательностные, чем комбинационные схемы. В комбинационной логике нет циклических путей и нет зависимостей состояния выхода от времени прохождения сигнала. Если на входы комбинационной логической схемы поданы определенные сигналы, то ее выход спустя некоторое время всегда установится в определенное корректное состояние. Однако, в последовательностных схемах с циклическими путями может появиться нежелательная нестабильность или гонки. Проверка таких схем требует много времени, и многие выдающиеся проектировщики делали подобные ошибки. Во избежание таких проблем разработчики разрывают циклические пути и добавляют в разрыв регистры. Это превращает схему в набор комбинационной логики и регистров. В регистрах содержится состояние системы, изменяющееся только по фронту тактового импульса. В этом случае говорят, что состояние синхронизировано с тактовым сигналом. Если период тактового сигнала достаточно большой, чтобы все входы регистров успели установиться до фронта следующего тактового импульса, то эффекты, связанные с гонками, устраняются. Следование правилу «всегда использовать регистры в обратной связи» приводит нас к формальному определению синхронной последовательностной схемы.

**Синхронные и асинхронные схемы**

Теоретически, из-за отсутствия временных ограничений, накладываемых на систему тактирующимися регистрами, при проектировании асинхронных схем разработчик обладает большей свободой, чем при проектировании синхронных. Таким же образом, как аналоговые схемы менее формализованы по сравнению с цифровыми, из-за того, что в аналоговых схемах могут использоваться произвольные напряжения, асинхронные схемы менее формализованы, чем синхронные, так как обратная связь в них может быть любой. Однако, оказывается, что синхронные схемы проектировать и использовать проще, чем асинхронные, так же как цифровые схемы проще проектировать, чем аналоговые. Несмотря на многолетнее научные исследования асинхронных схем, почти все современные цифровые схемы являются синхронными. Асинхронные схемы иногда используются для связи между собой систем с разными тактовыми сигналами или для считывания значений со входов в произвольное время, так же как аналоговые схемы необходимы для взаимодействия с реальным миром аналоговых (непрерывных) напряжений. Более того, среди разработок в области асинхронных схем есть действительно выдающиеся, некоторые из них могут также улучшить характеристики синхронных схем.

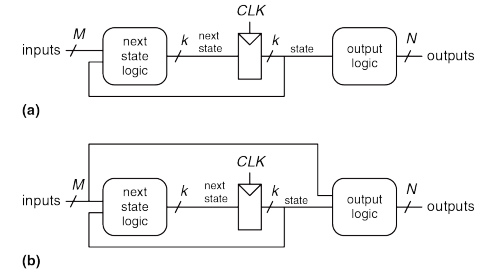
# Конечные автоматы. Пример проектирования конечного автомата.

**Конечные автоматы**

Они получили свое название из-за того, что схема с k-регистрами может находиться в одном из 2k , то есть в конечном числе, состояний. У КА М входов, N выходов и k бит состояний. На вход КА так же подается тактовый сигнал и, возможно, сигнал сброса. КА состоит из двух блоков комбинационной логики: логики перехода в следующее состояние и выходной логики, – и из регистра, в котором хранится текущее состояние. Существует два основных класса конечных автоматов, которые отличаются своими функциональными описаниями. В автомате Мура выходные значения зависят лишь от текущего состояния, в то время как в автомате Мили выход зависит как от текущего состояния, так и от входных данных. Конечные автоматы предоставляют систематический способ проектирования синхронных последовательностных схем по заданному функциональному описанию.

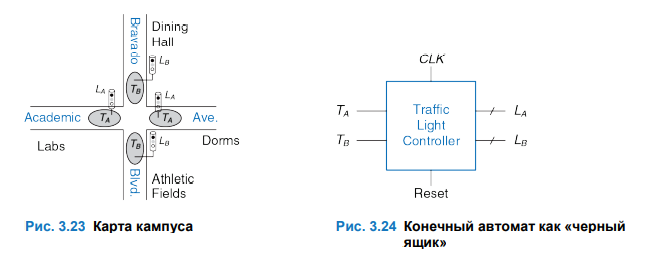
**Пример проектирования конечного автомата**

Для того чтобы проиллюстрировать процесс проектирования конечного автомата, рассмотрим проблему создания контроллера светофора для загруженного перекрестка в студенческом городке. Студенты-инженеры гуляют по Академической улице, на которой расположены учебные корпуса и общежитие. У них нет времени читать про конечные автоматы, и они не смотрят под ноги во время передвижения. Футболисты носятся между спортзалом и столовой по Беговой улице. Они гоняют мяч туда-сюда и тоже не смотрят под ноги. Несколько студентов уже получили серьезные травмы на перекрестке, и декан попросил Бена Битдидла установить светофор, пока не произошли инциденты с летальным исходом.

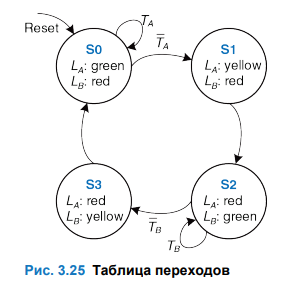
****

**Рис 3.22**

Бен решил справиться с проблемой с помощью конечного автомата. Он установил два датчика движения, TА и TБ, на Академической и Беговой улицах соответственно. Каждый датчик выдает единицу, если студенты присутствуют на улице и нуль, если никого нет. Он также установил два светофора для управления движением, LА и LБ. Каждый светофор получает входной цифровой сигнал, определяющий, каким светом он должен светить: красным, желтым или зеленым. Следовательно, у КА есть два входа, TА и TБ, и два выхода, LА и LБ. Перекресток с двумя светофорами и датчиками показан на Рис. 3.23. Бен подает тактовые импульсы раз в 5 секунд. По переднему фронту каждого импульса цвет светофора может измениться в зависимости от показаний датчиков движения. Также присутствует кнопка сброса, чтобы техники могли сбрасывать контроллер после подачи питания в известное исходное состояние. На Рис. 3.24 автомат изображен в виде «черного ящика».

****

Следующий шаг первокурсника – сделать набросок диаграммы переходов (или графа), показанный на Рис. 3.25, на котором приведены все возможные состояния системы и переходы между ними.

****

После сброса светофор горит зеленым на Академической улице и красным – на Беговой. Каждые 5 секунд контроллер анализирует движение и решает, что же делать дальше. Если движение присутствует на Академической улице, то цвет не меняется. Как только Академическая улица освобождается, на всех светофорах 5 секунд горит желтый, затем на Академической улице загорается красный, а на Беговой – зеленый. Аналогично, зеленый свет на Беговой улице сохраняется до тех пор, пока улица не станет свободной, затем светофор переключается на желтый, а затем – на красный. Кружки на диаграмме переходов обозначают состояния, а дуги со стрелками между ними – переходы между этими состояниями. Переходы осуществляются по переднему фронту тактового импульса. Мы не будем изображать тактовый сигнал на диаграмме, так как он всегда присутствует в синхронных логических схемах. Более того, тактовый сигнал лишь определяет, когда случится переход, тогда как диаграмма определяет, какой именно переход произойдет. Стрелка, обозначенная как Сброс, указывает на переход извне в состояние S0, отражая то, что система перейдет в это состояние сразу после сброса, независимо от того, в каком она была состоянии до этого. Если присутствует несколько стрелок, выходящих из некоторого состояния, то эти стрелки подписывают, чтобы показать, какой входной сигнал вызвал этот переход. Например, система находится в состоянии S0. Система останется в состоянии S0, если TA=1, и перейдет в состояние S1, если TA=0. Если из этого состояния выходит только одна стрелка, это означает, что такой переход произойдет вне зависимости от состояния входов. Например, из состояния S1 система всегда будет переходить в состояние S2, когда LA – красный, а LB – зеленый.

# Конечные автоматы. Кодирование состояний. Автоматы Мура и Мили.

**Конечные автоматы**

Они получили свое название из-за того, что схема с k-регистрами может находиться в одном из 2k , то есть в конечном числе, состояний. У КА М входов, N выходов и k бит состояний. На вход КА так же подается тактовый сигнал и, возможно, сигнал сброса. КА состоит из двух блоков комбинационной логики: логики перехода в следующее состояние и выходной логики, – и из регистра, в котором хранится текущее состояние. Существует два основных класса конечных автоматов, которые отличаются своими функциональными описаниями. В автомате Мура выходные значения зависят лишь от текущего состояния, в то время как в автомате Мили выход зависит как от текущего состояния, так и от входных данных. Конечные автоматы предоставляют систематический способ проектирования синхронных последовательностных схем по заданному функциональному описанию.

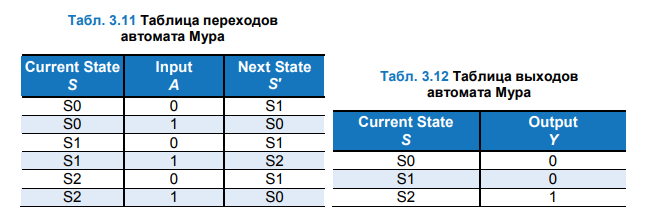
**Кодирование состояний**

Основная проблема заключается в том, как определить кодировку, которая потребует наименьшее количество элементов и приведет к наименьшим задержкам в схеме. К сожалению, простого способа найти самую лучшую кодировку не существует, кроме как перепробовать все возможные, что нерационально в случае, если число состояний велико. Однако зачастую возможно найти хорошую кодировку так, чтобы связанные состояния или выходы имели общие биты. При поиске набора возможных кодировок и выбора наиболее рациональной из них часто используются системы автоматизированного проектирования (САПР). Одно из важных решений в кодировании состояний – выбор между двоичным кодированием (00, 01, 10) и прямым кодированием (001, 010, 100), которое также называется кодированием «1 из N». При двоичном кодировании, как в примере с контроллером светофора, каждому состоянию ставится в соответствие двоичное число (номер этого состояния). Так как K двоичных чисел можно записать в log2K разрядах, системе с K состояниями нужно всего log2K битов состояния. В прямом кодировании для каждого состояния используется один бит состояния. Однако при использовании прямого кодирования схема определения следующего состояния и схема формирования выходных сигналов часто упрощается; таким образом, требуется меньше элементов. Наилучший выбор кодирования зависит от особенностей конкретного автомата.

**Автоматы Мура и Мили**

Вспомним, что автоматы Мили очень похожи на автоматы Мура, но значения на их выходах могут зависеть от значений на входах таким же образом, как они зависят от текущего состояния системы. Поэтому на диаграммах переходов для автоматов Миля значения выходов пишутся над стрелками. В блоке комбинационной логики, который вычисляет выходные значения, используются значения текущего состояния и входов, как показано на Рис. 3.32 (b).

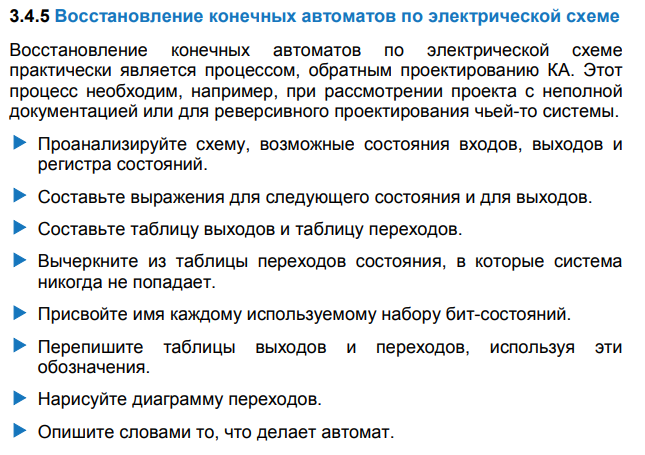
Простым способом запомнить разницу между двумя типами конечных автоматов состояний является тот факт, что у автомата Мура обычно больше (Moore – more) состояний, чем у автомата Мили, решающего ту же задачу.

****

# Декомпозиция конечных автоматов. Восстановление конечных автоматов по электрической схеме.

**Декомпозиция конечных автоматов**

Проектирование сложных конечных автоматов часто упрощается, если их можно разбить на несколько более простых автоматов, взаимодействующих друг с другом таким образом, что выход одних автоматов является входом других. Такое применение принципов иерархической организации и модульного проектирования называется декомпозицией конечных автоматов.

****

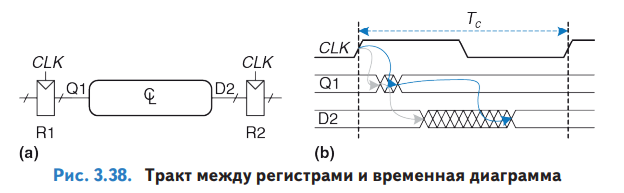
# Синхронизация последовательностных схем. Временные характеристики системы. Расфазировка тактовых сигналов. Метастабильность. Синхронизаторы.

**Синхронизация последовательных схем**

Вспомните, что триггер копирует сигнал с D-входа на Q-выход по переднему фронту тактового сигнала. Этот процесс называется фиксацией (sampling) D-сигнала по фронту тактового импульса. Поведение триггера корректно, если сигнал на D-входе стабилен (равен 0 или 1 и не изменяется) в течение переднего фронта тактового сигнала. Но что произойдет, если сигнал D не будет стабилен во время изменения тактового сигнала? Эта ситуация аналогична той, которая возникает при спуске затвора фотокамеры. Представьте, что вы пытаетесь снять прыжок лягушки с плавающего листа кувшинки в озеро. Если вы нажмете на спуск перед прыжком, то на фотографии вы увидите лягушку на листе кувшинки. Если вы нажмете на спуск после прыжка, то на фотографии будет рябь на воде. Но если вы нажмете на спуск во время прыжка, то на фотографии вы увидите смазанное изображение вытянутой вдоль направления прыжка лягушки. Одной из характеристик фотокамеры является апертурное время, в течение которого фотографируемый объект должен быть неподвижен, чтобы на фотографии сформировалось его резкое изображение. Аналогично, последовательный элемент имеет апертурное время до и после фронта тактового сигнала, в течение которого его информационные входные сигналы должны быть стабильными, чтобы на выходе триггера сформировался корректный сигнал.

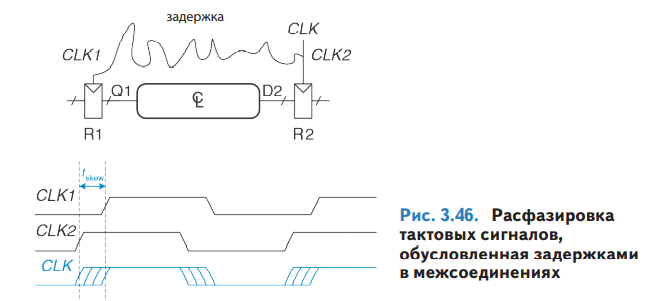
**Временные характеристики системы**

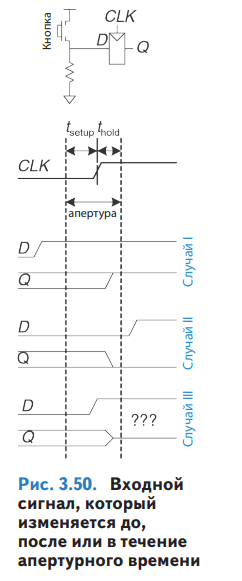
Периодом тактового сигнала или длительностью цикла синхронизации, Tc , называется промежуток времени между передними фронтами последовательных тактовых импульсов. Обратная величина, fc = 1/Tc , называется тактовой частотой. Увеличение тактовой частоты без изменения остальных параметров схемы приводит к увеличению ее производительности. Частота измеряется в герцах (Гц), или в циклах за одну секунду: 1 мегагерц (МГц) = 106 Гц и 1 гигагерц (ГГц) = 109 Гц. На Рис. 3.38 (a) показана характерная структура тракта обработки информации синхронной последовательной схемы, для которой мы рассчитаем период тактового сигнала. По переднему фронту тактового импульса на выходе регистра R1 формируется выходной сигнал (или сигналы) Q1. Эти сигналы поступают на вход блока комбинационной логики, выходные сигналы этого блока поступают на вход (или входы) D2 регистра R2. Как показано на Рис. 3.38 (b), выходной сигнал блока может начать изменяться не ранее окончания времени реакции после завершения изменения его входного сигнала и принимает окончательное значение спустя максимальное время задержки распространения от момента установления входного сигнала. Серые стрелки показывают минимальную задержку с учетом R1 и комбинационной логики, а синие – максимальную задержку распространения в тракте регистр R1 – комбинационная логика. Мы проанализируем временные ограничения с учетом времен предустановки и удержания второго регистра, R2.



**Расфазировка тактовых сигналов**

В предыдущих разделах предполагалось, что тактовые импульсы поступают на все регистры в одно и то же время. В действительности существует некоторый разброс этого времени. Эта неодновременность фронтов называется расфазировкой. Например, длина проводников, по которым тактовые сигналы поступают на разные регистры, может быть разной, это приводит к разным временам задержки, как показано на Рис. 3.46.



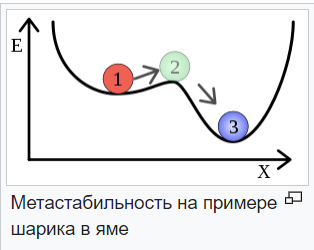
**Метастабильность**

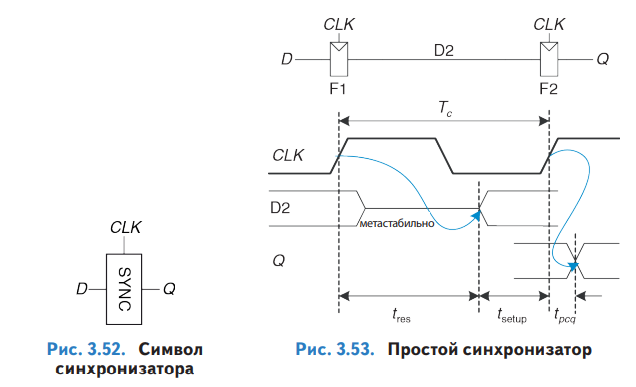
**Метастабильное состояние** (от [греч.](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B5%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) μετα «через» и [лат.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%82%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *stabilis* «устойчивый») — состояние квазиустойчивого [равновесия](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B2%D0%BD%D0%BE%D0%B2%D0%B5%D1%81%D0%B8%D0%B5) [физической системы](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B7%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), в котором система может находиться длительное время.

Что такое метастабильное состояние, может быть понятно из приведенного справа рисунка:

* состояние 1: **метастабильное** — состояние, стабильность которого сохраняется при не очень больших возмущениях;
* состояние 2: **нестабильное** — состояние, стабильность которого нарушается при сколь угодно малых возмущениях;
* состояние 3: **стабильное** — состояние, стабильность которого сохраняется при больших возмущениях.

**Синхронизаторы**

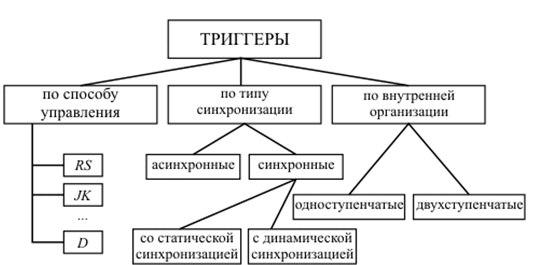
Наличие асинхронных входов цифровой системы, которые принимают информацию из внешнего мира, неизбежно. Например, сигналы, которые формирует человек, асинхронны. Такие асинхронные входы, если к ним относиться небрежно, могут привести к появлению метастабильных состояний в системе, что приведет к ее непредсказуемым отказам, которые крайне сложно отследить и исправить. При наличии асинхронных входов проектировщик системы должен обеспечить достаточно малую вероятность появления метастабильных напряжений. Смысл слова «достаточно» зависит от контекста. Для сотового телефона, вероятно, один отказ за 10 лет допустим, потому что пользователь может всегда выключить и включить телефон, если он «зависнет». Для медицинского прибора более предпочтительным является один отказ за предполагаемое время существования вселенной (1010 лет). Чтобы гарантировать корректность логических уровней, все асинхронные входы должны пройти через синхронизаторы. Синхронизатор, как показано на Рис. 3.52, является устройством, на вход которого поступает асинхронный сигнал D и тактовый сигнал CLK. За ограниченное время он формирует выходной сигнал Q, который с очень высокой вероятностью имеет корректный логический уровень. Если вход D стабилен в течение апертурного времени, то выход Q должен принять значение входа. Если D изменяется в течение апертурного времени, то Q может принять значение 0 или 1, но не должен быть метастабильным. На Рис. 3.53 показано, как из двух триггеров можно построить простой синхронизатор. Триггер F1 фиксирует значение входного сигнала D по переднему фронту тактового сигнала CLK. Если D изменяется в апертурное время, его выход D2 на некоторое время может стать метастабильным. Если период тактового сигнала достаточно велик, то с высокой вероятностью до конца периода D2 придет к корректному логическому**** уровню. Триггер F2 затем фиксирует D2, который теперь стабилен, и формирует корректный выходной сигнал.



# Типы триггеров. Классификация триггеров. RS-триггер на элементах И-НЕ и ИЛИ–НЕ. T-, JK-, D-триггеры.

**Триггер**– электронная схема, обладающая двумя устойчивыми состояниями. Переход из одного устойчивого состояния в другое происходит скачкообразно под воздействием управляющих сигналов. При этом также скачкообразно изменяется уровень напряжения на выходе триггера.

Триггеры служат основой для построения регистров, счетчиков и других элементов, обладающих функцией хранения. Главной частью любого триггера является запоминающая ячейка (ЗЯ).

**Типы триггеров**

Триггер называется синхронным, если его таблица переходов хотя бы по одному управляющему входу реализуется под воздействием синхронизирующего сигнала.

Основу синхронного одноступенчатого триггера составляет рассмотренная выше запоминающая ячейка (элементы 1, 2). Комбинационная схема преобразует управляющие сигналы триггера, а также, для некоторых типов триггеров, сигналы Q и с выходов ЗЯ в сигналы S и R на входах запоминающей ячейки.

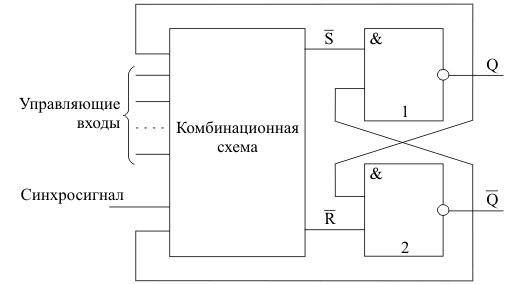
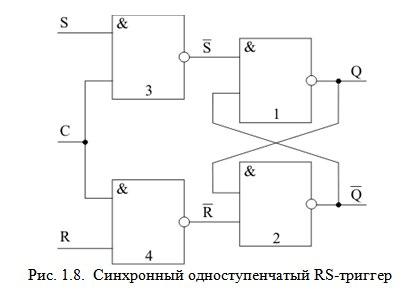
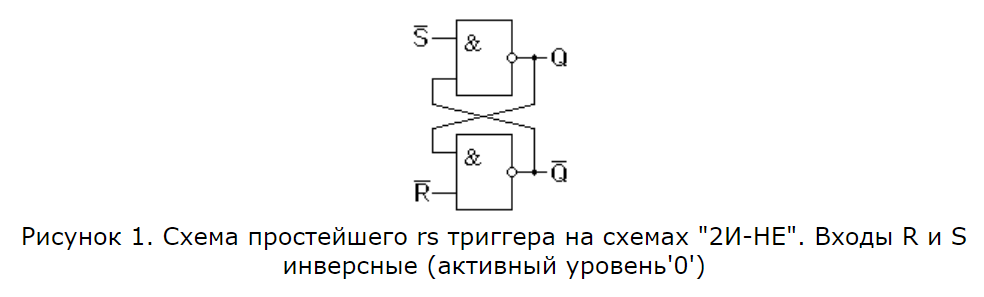
Обобщенная схема синхронного одноступенчатого триггера:

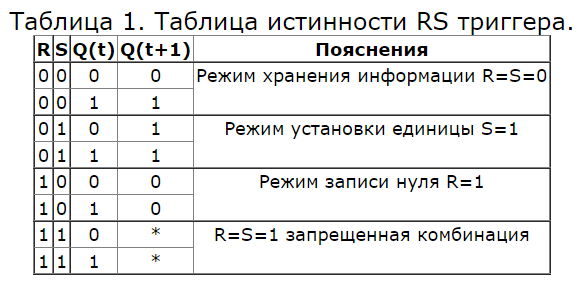
Схема синхронного одноступенчатого RS-триггера:



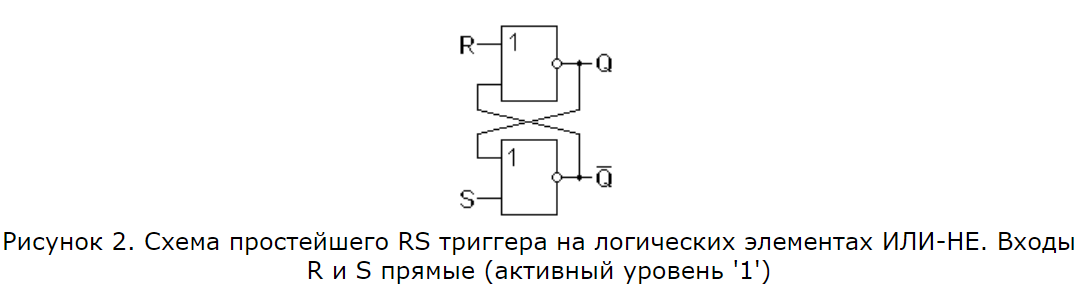


Рассмотрим принцип работы RS триггера, выполненный по изображенной на рисунке 1 схеме подробнее. Пусть на входы R и S подаются единичные потенциалы. Если на выходе верхнего логического элемента "2И-НЕ" Q присутствует логический ноль, то на выходе нижнего логического элемента "[2И-НЕ](https://digteh.ru/digital/TTL/)" появится логическая единица. Эта единица подтвердит логический ноль на выходе Q. Если на выходе верхнего логического элемента "2И-НЕ" Q первоначально присутствует логическая единица, то на выходе нижнего логического элемента "2И-НЕ" появится логический ноль. Этот ноль подтвердит логическую единицу на выходе Q. То есть при единичных входных уровнях схема RS триггера работает точно так же как и схема на инверторах.

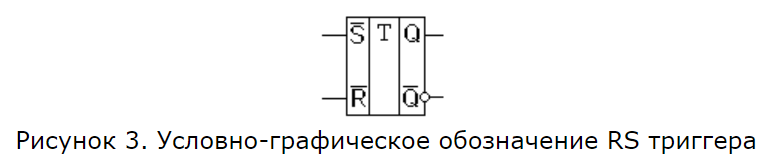
Подадим на вход S нулевой потенциал. Согласно таблице истинности логического элемента "И-НЕ" на выходе Q появится единичный потенциал. Это приведёт к появлению на инверсном выходе триггера нулевого потенциала. Теперь, даже если снять нулевой потенциал с входа S, на выходе триггера останется единичный потенциал. То есть мы записали в триггер логическую единицу.

Точно так же можно записать в RS-триггер и логический ноль. Для этого следует воспользоваться входом R. Так как активный уровень на входах оказался нулевым, то эти входы — инверсные. Составим таблицу истинности RS триггера. Входы R и S в этой таблице будем использовать прямые, то есть и запись нуля, и запись единицы будут осуществляться единичными потенциалами (таблица 1).

RS триггер можно построить и на логических элементах "ИЛИ". Его схема приведена на рисунке 2. Принцип работы RS триггера, собранный на логических элементах "ИЛИ" будет точно таким же, как и рассмотренный ранее. Единственное отличие в работе этой схемы по сравнению с предыдущей схемой RS триггера будет заключаться в том, что сброс и установка триггера будет производиться единичными логическими уровнями. Эти особенности связаны с принципами работы инверсной логики, которые рассматривались ранее.

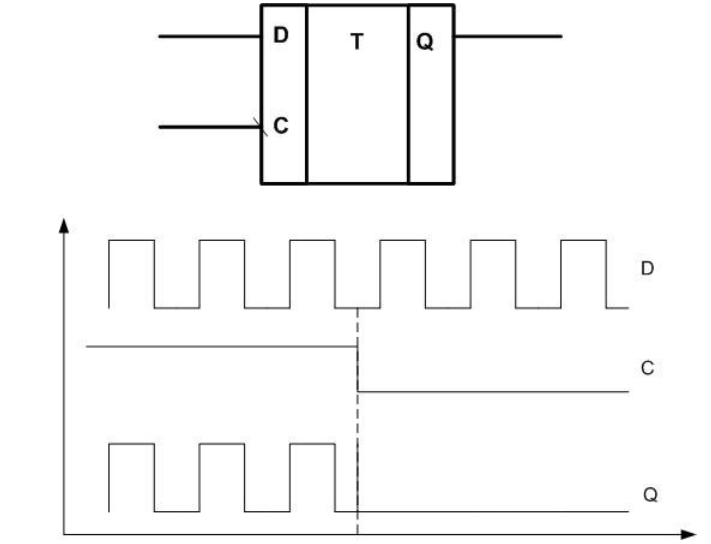


Так как RS триггер при реализации его на логических элементах "И" и "ИЛИ" работает одинаково (его принцип работы от схемы не зависит), то и условно-графическое изображение на принципиальных схемах тоже одинаково. [Условно-графическое изображение](https://digteh.ru/digital/DefDigCh.php) RS триггера приведено на рисунке 3.

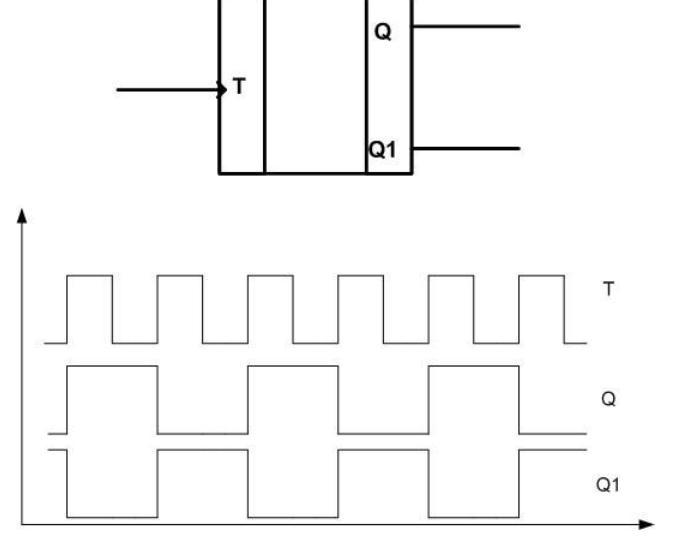


D-триггер («прозрачный триггер», «защелка», latch) относится к категории синхронных устройств, тактируемых по входу С. Также имеется вход для данных D (Data). По функциональным возможностям устройство относится к триггерам с приёмом информации по одному входу.

Пока на входе для синхронизации присутствует логическая единица, сигнал на выходе Q повторяет сигнал на входе данных (режим прозрачности). Как только уровень строба перейдет в состояние 0, на выходе Q уровень останется тем же, что был в момент перепада (защелкнется). Так можно зафиксировать входной уровень на входе в любой момент времени. Также существуют D-триггеры с тактированием по фронту. Они защёлкивают сигнал по положительному перепаду строба.

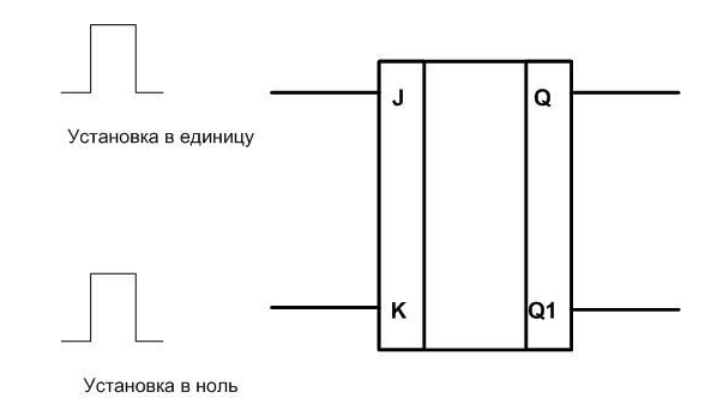


T-триггер относится к классу счётных бистабильных элементов. Логика его работы проста – он изменяет своё состояние каждый раз, когда на его вход приходит очередная логическая единица. Если на вход подать импульсный сигнал, выходная частота будет в два раза выше входной. На инверсном выходе сигнал будет противофазен прямому.



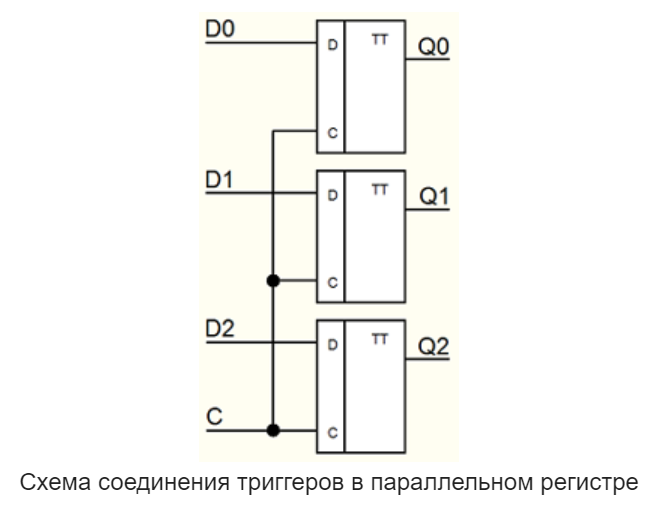
JK-триггеры

Этот бистабильный элемент относится к категории универсальных. Он может управляться раздельно по входам. Логика работы JK-триггера похожа на работу RS-элемента. Для установки выхода в единицу используется вход J (Job). Появление высокого уровня на выводе K (Keep) сбрасывает выход в ноль. Принципиальным отличием от RS-триггера является то, что одновременное появление единиц на двух управляющих входах не является запретным. В этом случае выход элемента меняет свое состояние на противоположное.



# Параллельные и последовательные регистры. Отличия в обозначения цифровых элементов в разных стандартах.

**Параллельный регистр**

****В данном типе регистров триггеры соединены параллельно, то есть каждый внутренний триггер имеет свой вход D и свой выход Q, которые не зависят от других триггеров, а также вход С, который называется **тактовым входом** и для всех входящих в регистр триггеров он является общим. Параллельные регистры бывают двух типов:

* **тактируемые регистры**, которые срабатывают по **фронту сигнала** управления (вход С);
* **стробируемые регистры**, которые срабатывают по **уровню входного сигнала** (вход С);

Наибольшее распространение получили тактируемые параллельные регистры.

**Регистры, срабатывающие по фронту сигнала**

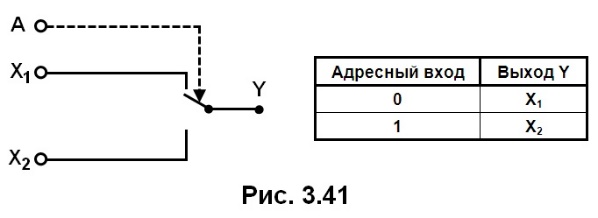
Данные типы регистров практически ничем не отличаются от D триггеров. Напомню, что на выходе Q устанавливается такой уровень напряжения, который был на входе D, в момент положительного фронта тактового сигнала на входе С. Так как регистр состоит из нескольких D триггеров, то и количество (4,6,8,16) запоминаемых сигналов (бит) у регистра больше.

**Регистры, срабатывающие по уровню сигнала**

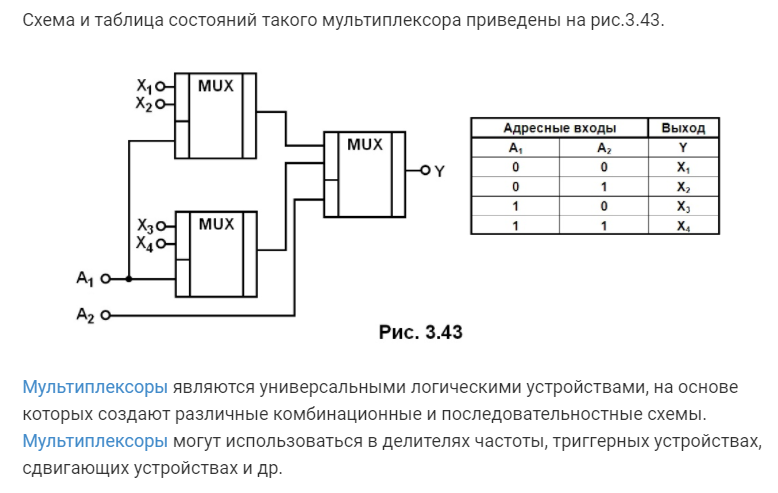
Данные типы регистров являются как бы гибридом между буфером и регистром (ещё такие регистры называют **регистры-защёлки**, англ. Latch). Принцип работы данных регистров состоит в следующем: когда на **стробирующем входе** высокий логический уровень, сигнал **поступает с входов** на выходы регистра, а если на стробирующем входе низкий уровень сигнала, регистр переходит в **режим хранения** последнего из пропущенных значений входного сигнала. Регистры-защелки не очень распространены из-за своего специфического характера работы, но в некоторых случаях удобнее, **регистров срабатывающих по фронту сигнала** на тактовом входе.

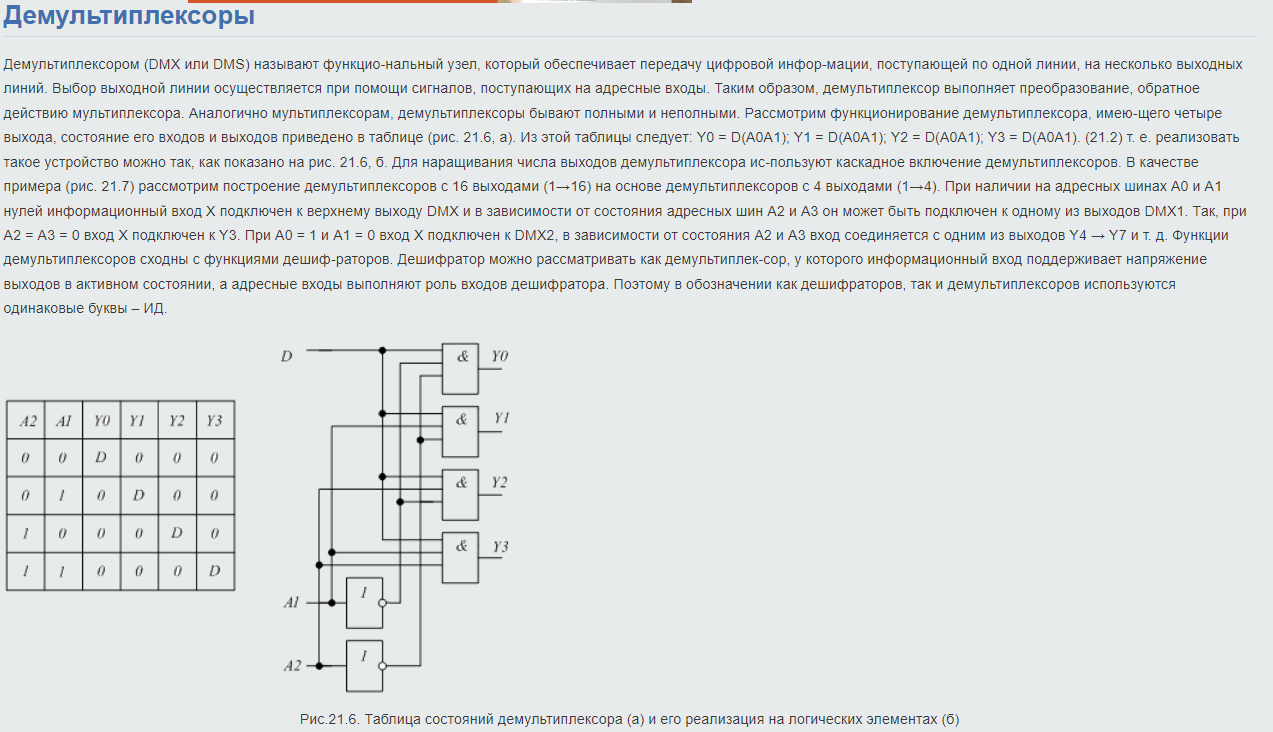
# Мультиплексоры и демультиплексоры. Отличия в обозначения цифровых элементов в разных стандартах.

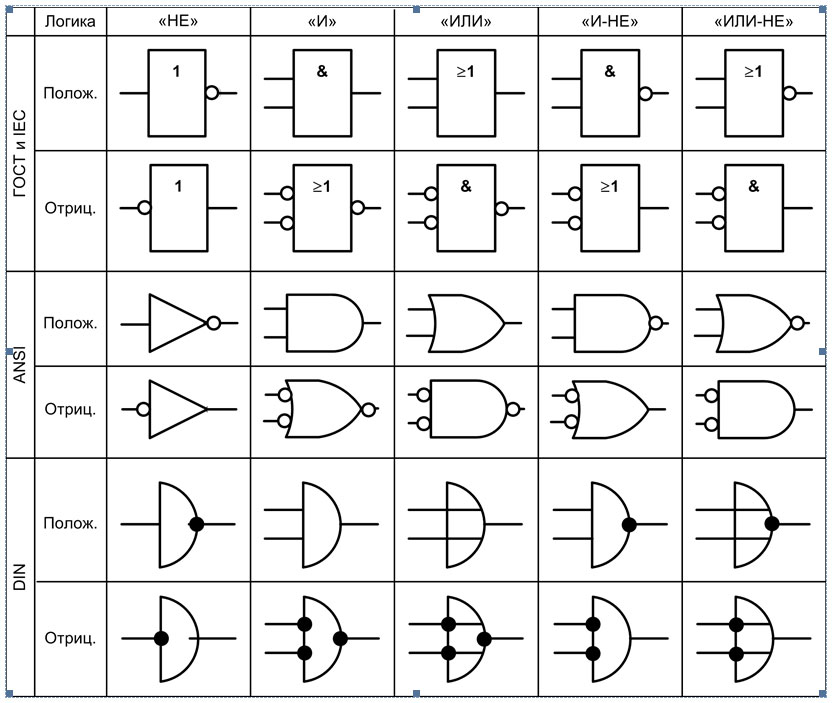
Мультиплексором — называют комбинационное устройство, обеспечивающее передачу в желаемом порядке цифровой информации, поступающей по нескольким входам на один выход. Мультиплексоры обозначают через MUX (от англ. multiplexor), а также через MS (от англ. multiplex or selector).



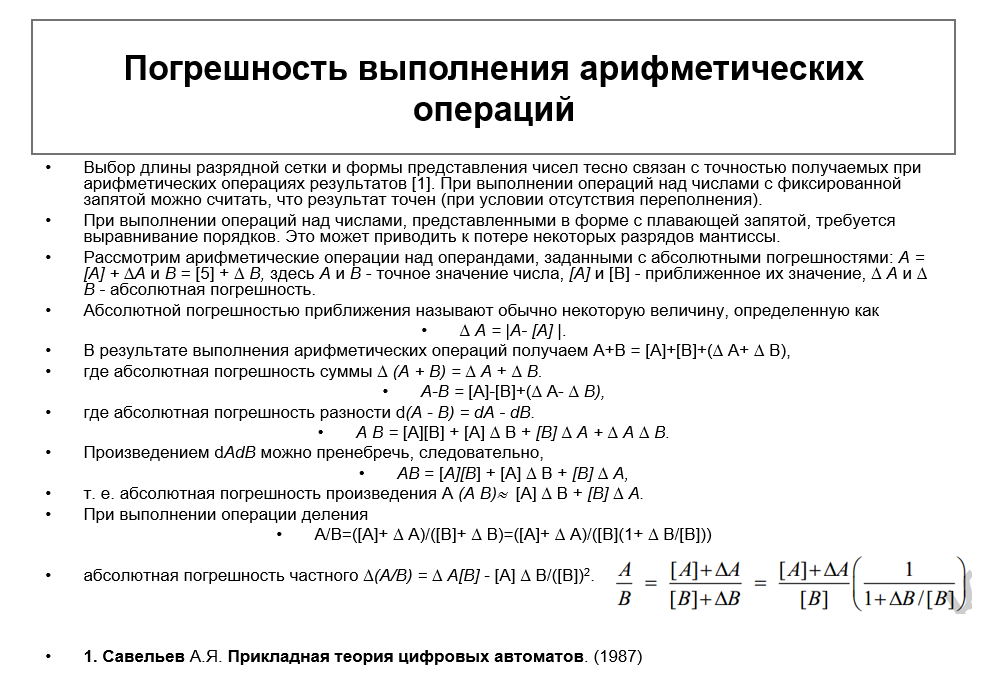
Сигналы на адресных входах определяют, какой конкретно информационный канал подключен к выходу. Если между числом информационных входов n и числом адресных входов m действует соотношение n = 2m, то такой мультиплексор называют полным. Если n< 2ь, то мультиплексор называют неполным.



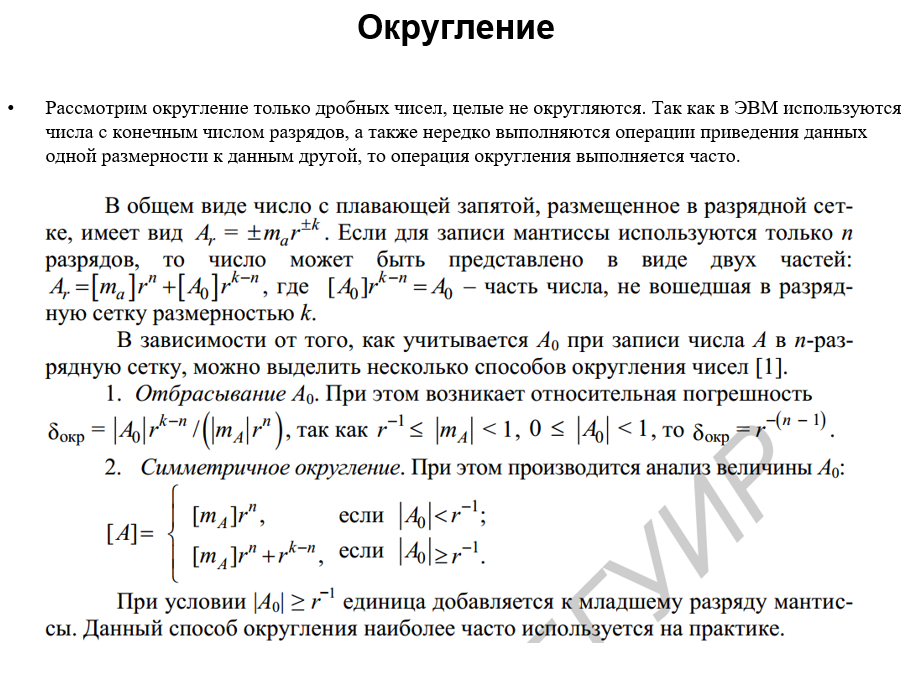
****

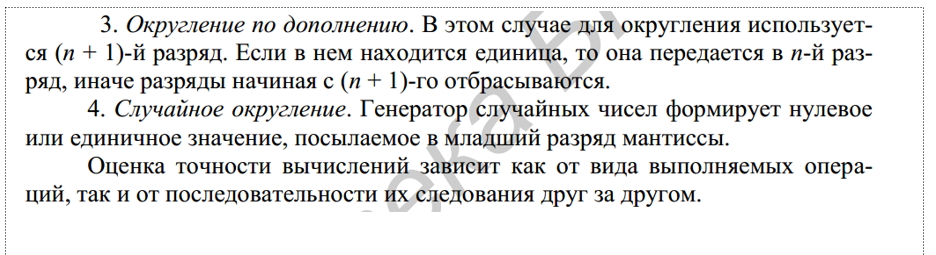


# Погрешность метаматематических операций в цифровых системах. Способы оценки. Округление.

****

****

****

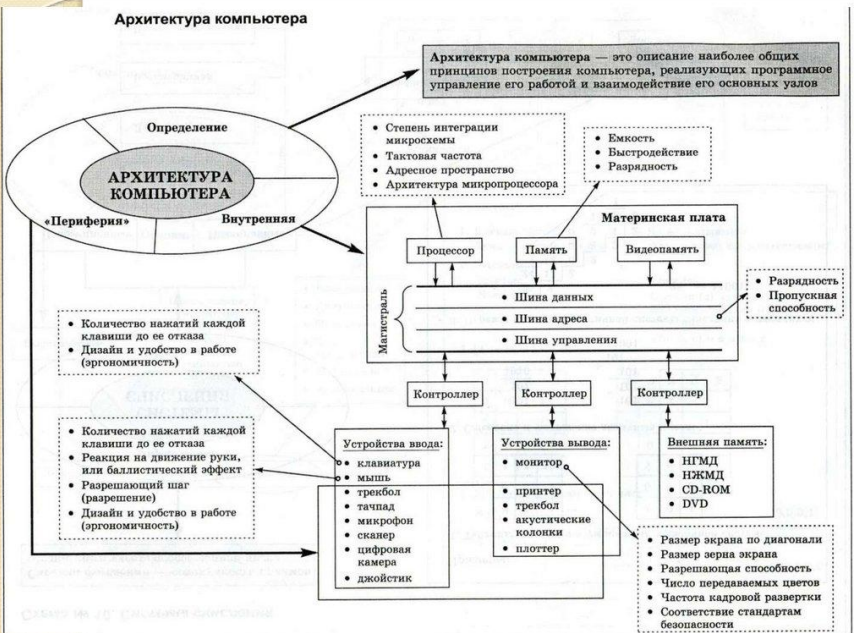
****

# Архитектура процессора. Основные компоненты. Способы классификации. Много уровневая организация. Контроллеры ввода-вывода.

**Архитектура процессора**

Архитекту́ра проце́ссора — количественная составляющая компонентов микроархитектуры вычислительной машины (процессора компьютера) (например, регистр флагов или регистры процессора), рассматриваемая IT-специалистами в аспекте прикладной деятельности.

**Основные компоненты**



**Способы классификации**

Имеются различные классификации архитектур процессоров как по организации (например, по количеству и сложности отдельных команд: RISC, CISC; по возможности доступа команд к памяти[1]), так и по назначению (например, специализированные графические, математические или предназначенные для цифровой обработки сигналов)

**Многоуровневая организация**



**Контроллеры ввода-вывода**

1)Super I/O (англ. Super Input/output) — название класса сопроцессоров, которые начали использоваться после 1980-х годов на материнских платах IBM PC-совместимых компьютеров. Super I/O объединяет интерфейсы различных низкоскоростных устройств.

Как правило, включает в себя следующие функции:

• контроллер дисковода гибких дисков (floppy);

• контроллер параллельного порта (LPT-порт);

• контроллер последовательных (COM) портов и портов клавиатуры и мыши (PS/2).

• игровой (MIDI или джойстик) или инфракрасный порты.

• контроллер Ethernet

# RISK, CISK, MISC, VLIW. Отличительные особенности, сфера применения. Что такое Spectre и Meltdown.



**RISK**

RISC (англ. Reduced Instruction Set Computer — «компьютер с сокращённым набором команд») — архитектура процессора, в котором быстродействие увеличивается за счёт упрощения инструкций: их декодирование становится более простым, а время выполнения — меньшим. Первые RISC-процессоры не имели даже инструкций умножения и деления и не поддерживали работу с числами с плавающей запятой (RISC быстрее CISC)

**CISK**

CISC (англ. Complex Instruction Set Computer — «компьютер с полным набором команд») — тип процессорной архитектуры, в первую очередь, с нефиксированной длиной команд, а также с кодированием арифметических действий в одной команде и небольшим числом регистров, многие из которых выполняют строго определенную функцию (В CISC процессорах одна команда может быть заменена ей аналогичной, либо группой команд, выполняющих ту же функцию)

**MISC**

MISC (англ. Minimal Instruction Set Computer — «компьютер с минимальным набором команд») - ещё более простая архитектура, используемая в первую очередь для ещё большего уменьшения итоговой цены и энергопотребления процессора. Используется в IoT-сегменте и недорогих компьютерах, например, роутерах. (“спекулятивное исполнение команд” - это выполнение команды до того, как станет известно, понадобится эта команда или нет)

**VLIW**

VLIW (англ. Very Long Instruction Word — «очень длинная машинная команда») — архитектура процессоров с несколькими вычислительными устройствами. Характеризуется тем, что одна инструкция процессора содержит несколько операций, которые должны выполняться параллельно. По сути является архитектурой CISC со своим аналогом спекулятивного исполнения команд, только сама спекуляция выполняется во время компиляции, а не во время работы программы, из-за чего уязвимости Meltdown и Spectre невозможны для этих процессоров.

**Что такое Spectre и Meltdown**

Spectre — группа аппаратных уязвимостей, ошибка в большинстве современных процессоров, имеющих спекулятивное выполнение команд (англ.)рус. и развитое предсказание ветвлений, позволяющих проводить чтение данных через сторонний канал в виде общей иерархии кэш-памяти. Затрагивает большинство современных микропроцессоров, в частности, архитектур х86/x86\_64 (Intel и AMD) и некоторые процессорные ядра ARM.

Meltdown — аппаратная уязвимость категории утечка по стороннему каналу, обнаруженная в ряде микропроцессоров, в частности, производства Intel и архитектуры ARM. Meltdown использует ошибку реализации спекулятивного выполнения команд (англ.) рус. в некоторых процессорах Intel и ARM (но не AMD), из-за которой при спекулятивном выполнении инструкций чтения из памяти процессор игнорирует права доступа к страницам.

# Сравнительная характеристика архитектур. В чем преимущества. Преимущества RISC. Какова проблема лицензирования архитектур.

**Сравнительная характеристика архитектур**



**Преимущества RISC**

RISC проще, легче оптимизируется, соответственно быстрее, меньше потребление, простота наращивания и отладки, инструкции фиксированной длины, не нужно делать выравнивание, проще работа с памятью, более богатая регистровая архитектура, легче делать 32/64/128 разрядов (далее везде)

**Какова проблема лицензирования архитектур**

Конкуренция

# Виртуальные архитектуры. Команды (инструкции), предназначение, виды. Тактирование процессоров. Выполнение инструкций. Поток инструкций

**Виртуальные архитектуры**

Магические JAR-файлы, которые можно запустить на любой машине - это пример виртуальной JVM-архитектуры, которая, по сути, эмулируется на целевой реальной машине. Поэтому достаточно JVM-машины для целевой архитектуры для запуска на ней любой Java-программы. Другим примером виртуальной архитектуры является .NET CIL.

Из минусов виртуальных архитектур можно выделить меньшую производительность по сравнению с реальными архитектурами. Однако большим плюсом будет кроссплатформенность.

**Команды (инструкции), предназначение, виды**

Инструкция - это не что иное, как действие, которое мы отправляем процессору. Инструкции могут быть арифметическими операциями с различными типами данных, такими как с плавающей запятой, целыми числами, вектором, скаляром, логическими операциями, операциями перемещения данных, операциями перемещения битов (где бит изменяет положение), операциями перехода и т. д.

Они бывают нескольких типов:

• Арифметические: сложение, вычитание, умножение и т. д.

• Логические: И (логическое умножение/конъюнкция), ИЛИ (логическое суммирование/дизъюнкция), отрицание и т. д.

• Информационные: move, input, outptut, load и store.

• Команды перехода: goto, if ... goto, call и return.

• Команда останова: halt.

**Тактирование процессоров**

Быстродействие компьютера определяется тактовой частотой его процессора. Тактовая частота — количество тактов (соответственно и исполняемых команд) за секунду.

Частота нынешних процессоров измеряется в ГГц (Гигагерцы). 1 ГГц = 10⁹ Гц — миллиард операций в секунду.

Чтобы уменьшить время выполнения программы, нужно либо оптимизировать (уменьшить) её, либо увеличить тактовую частоту. У части процессоров есть возможность увеличить частоту (разогнать процессор), однако такие действия физически влияют на процессор и нередко вызывают перегрев и выход из строя.

**Выполнение инструкций**

Инструкции хранятся в ОЗУ в последовательном порядке. Для гипотетического процессора инструкция состоит из кода операции и адреса памяти/регистра. Внутри управляющего устройства есть два регистра инструкций, в которые загружается код команды и адрес текущей исполняемой команды.

Т.е., независимо от того, какой процессор использует наша система, все они читают двоичный код определенным образом, соответствующим своему семейству. Что они делают, так это берут определенное количество бит двоичного кода, который они выполняют, и интерпретируют их значение в соответствии с его расположением. Каждая инструкция кодируется следующим образом: первые цифры соответствуют коду инструкции и способу его выполнения, а последние биты - это сами данные или место, где находятся данные, на которых мы хотим выполнить инструкцию.

**Поток инструкций**

Современные процессоры могут параллельно обрабатывать несколько команд. Пока одна инструкция находится в стадии декодирования, процессор может успеть получить другую инструкцию. Однако такое решение подходит только для тех инструкций, которые не зависят друг от друга. Если процессор многоядерный, это означает, что фактически в нём находятся несколько отдельных процессоров с некоторыми общими ресурсами, например кэшем.

# Регистр процессора: предназначение, виды. Шины: предназначение, виды.. Кэш: предназначение, виды.

**Регистр процессора**

Регистры - это память, ближайшая к существующему процессору и, следовательно, самая быстрая; Это очень маленькие блоки памяти, которыми можно управлять напрямую с помощью блока управления процессора. Они используются для выполнения всех видов общих задач, а не только для выполнения арифметических операций.

Наиболее распространенные регистры в процессоре независимо от его ISA:

• Регистры типа аккумулятора : используется для арифметических операций. Каждое семейство имеет разное количество записей типа аккумулятора.

• Регистры доступа к памяти : содержат адрес памяти данных, к которым мы хотим получить доступ из ОЗУ.

• Регистры данных в или из памяти : Содержат данные, скопированные из памяти (чтение) или для записи по определенному адресу памяти (запись).

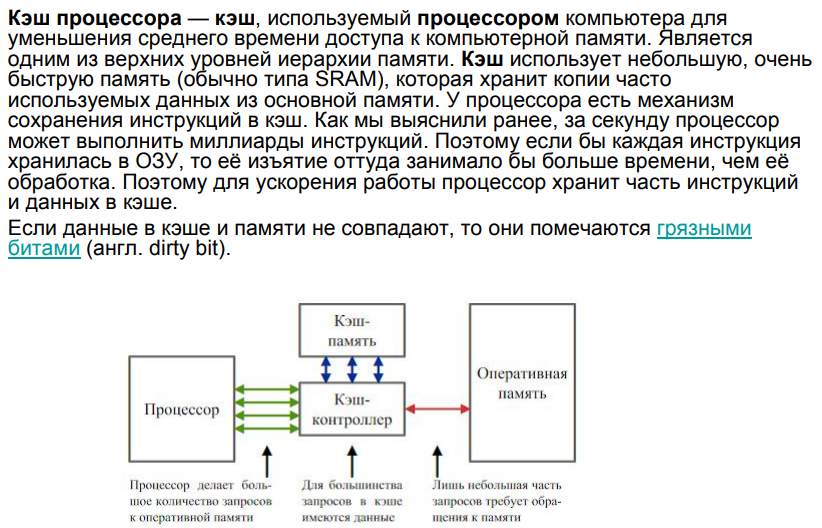
• Регистры общего назначения : это регистры памяти без специальной утилиты, которые служат для хранения данных, которые должны быть вызваны как можно быстрее.

• Счетчик команд : указывает следующую инструкцию для выполнения; Команды перехода изменяют их, когда вы хотите получить доступ не к следующей инструкции, а к другой части программы. В каждом полном командном цикле адрес памяти увеличивается на 1 и связывается с адресной шиной процессора.

**Шины**



**Кэш**



# Что такое суперскалярная архитектура. Ее особенности. Предсказатели переходов. Иерархия памяти. Ветвление (посл. пункт ???)

**Суперскалярная архитектура**

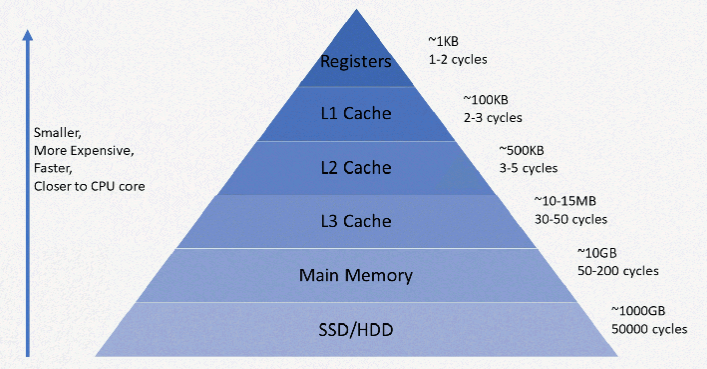
Это означает, что в любой момент времени процессор одновременно выполняет на каждом этапе конвейера множество инструкций. Он может также ожидать ещё сотни других, чтобы начать их выполнение, и для того, чтобы иметь возможность одновременного выполнения нескольких инструкций внутри процессоров есть несколько копий каждого этапа конвейера. Если процессор видит, что к выполнению готовы две инструкции, и между ними нет зависимости, то он не ждёт, пока они завершатся по отдельности, а выполняет их одновременно.

**Предсказатели переходов**

Инструкции переходов (ветвлений) схожи с конструкциями «if» для процессора. Один набор инструкций выполняется, если условие истинно, а другой — если оно ложно. Например, нам нужно сравнить два числа, и если они равны, выполнить одну функцию, а если не равны, то выполнить другую. Эти инструкции ветвления применяются чрезвычайно часто и могут составлять примерно 20% всех инструкций в программе.

**Иерархия памяти**

Кэш L1 — самый маленький и быстрый, L2 находится посередине, а L3 — самый крупный и медленный из всех кэшей. Выше кэшей в иерархии находятся мелкие регистры, хранящие во время вычислений единственное значение данных. По порядку величин эти регистры являются самыми быстрыми устройствами хранения в системе. Когда компилятор преобразует высокоуровневую программу в язык ассемблера, он определяет наилучший способ использования этих регистров.



# Что такое гетерогенные вычисления. FPGA-акселератор? Сфера применения. Перспективные направления развития вычислительных систем. (посл. пункт ???)

**Гетерогенные вычисления**

Эта методика заключается во включении в одну систему множества различных вычислительных элементов. Большинство из нас пользуется преимуществами такого подхода в виде отдельных GPU в компьютерах. Центральный процессор очень гибок и может с приличной скоростью выполнять широкий диапазон вычислительных задач. С другой стороны, GPU спроектированы специально для выполнения графических вычислений, например, перемножения матриц. Они очень хорошо с этим справляются и на порядки величин быстрее ЦП в подобных видах инструкций. Перенеся часть графических вычислений с ЦП на GPU, мы можем ускорить расчёты. Любой программист может оптимизировать ПО, изменив алгоритм, но оптимизировать оборудование гораздо сложнее.

**FPGA-акселератор**

В отличие от традиционных вычислительных элементов наподобие ЦП и GPU, имеющих фиксированную внутреннюю архитектуру, FPGA гибки. Это почти программируемое оборудование, которое можно настраивать в соответствии с нуждами компании.

Если кому-то нужно распознавание изображений, то он реализует эти алгоритмы в «железе». Если кто-то хочет симулировать работу новой аппаратной архитектуры, то перед изготовлением её можно протестировать на FPGA. FPGA обеспечивает большую производительность и энергоэффективность, чем GPU, но всё равно меньше, чем у ASIC (application specific integrated circuit — интегральная схема специального назначения). Другие компании, например, Google и Nvidia, разрабатывают отдельные ASIC машинного обучения для ускорения распознавания и анализа изображений.