|  |
| --- |
| **Вопрос 1. Системы счисления. Перевод чисел из одной системы счисления в другую. Метод преобразования с использованием весов разрядов.** |

**Системы счисления**

Система счисления - это способ записи чисел с помощью заданного набора специальных знаков (цифр).

В своей повседневной деятельности человек использует различные **системы счисления**, к которым относятся десятичная, римская, система исчисления времени и т. д. Все их можно подразделить **на позиционные и непозиционные**. В **непозиционных** системах счисления «доля» цифры в ее количественном измерении не зависит от местоположения данной цифры в записи этого числа. Типичным примером такой системы счисления является римская. В ней используются цифры:

I V X L C D M – римские цифры

1 5 10 50 100 500 1000 – их десятичные эквиваленты

ТО ЕСТЬ: Непозиционная система счисления – система счисления, в которой для обозначения чисел вводятся специальные знаки, количественное значение которых («вес» символа) всегда одинаково и не зависит от их места в записи числа.

*Пример.*

Значение числа МММСМLIХ определяется как сумма:

1000 + 1000 + 1000 + (1000 *–* 100) + 50 + (10 *–* 1),

что соответствует десятичному эквиваленту 3959.

Количественная оценка числа, записанного в **позиционной** системе счисления, определяется как сумма произведений значения цифр, составляющих запись числа, умноженных на вес позиции, в которой располагается цифра.

Примером такой системы счисления является широко используемая **десятичная система счисления. (а еще восьмеричная, двоичная, шестнадцатеричная)**

*Пример.*

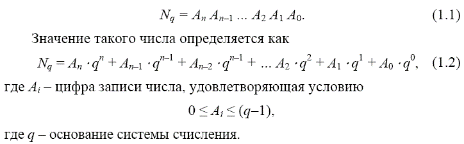
Количественная оценка десятичного числа 395910 определяется как

3 ⋅ 1000 + 9 ⋅ 100 + 5 ⋅ 10 + 9 ⋅ 1,

где 1000, 100, 10, 1 – соответственно вес четвертого, третьего, второго, первого разрядов записи оцениваемого числа.

***Десятичная система счисления***является также системой с равномерно распределенными весами, которые характеризуются тем, что соотношение весов двух любых соседних разрядов имеет для такой системы одинаковое значение. Это соотношение называется основанием системы счисления, которое далее будем обозначать как «*q*».

Общая запись числа в системе с равномерно распределенными весами имеет вид



Запись числа *N* в виде (1.1) называется **кодированной**, а запись в форме (1.2) – **расширенной**.

Помимо *q* = 10 (десятичная система счисления), возможны другие значения для основания системы счисления:

− двоичная система счисления;

− восьмеричная система счисления;

− шестнадцатеричная система счисления и т. д.

В различных системах счисления в качестве цифр используются обозначения соответствующих цифр десятичной системы счисления – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а в случае, когда десятичных цифр «не хватает» (для систем счисления с основанием *q*, большим чем 10), для цифр, превышающих 9, вводятся дополнительные обозначения, например, для *q* = 16 это будут обозначения А, В, C, D, E, F, которые соответствуют шестнадцатеричным цифрам (десятичные эквиваленты их равны соответственно 10, 11, 12, 13, 14, 15).

**Перевод чисел из одной системы счисления в другую**

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.

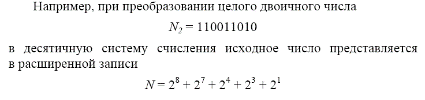
− деления (умножения) на новое основание.

− с использованием особого соотношения заданной и искомой систем счисления.

**Метод преобразования с использованием весов разрядов**

Метод преобразования с использованием весов разрядов записи числа в исходной и в искомой системах **предполагает применение расширенной записи числа** (1.2) в некоторой системе счисления. 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.



и 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.

|  |
| --- |
| **Вопрос 2. Перевод чисел из одной системы счисления в другую. Метод деления (умножения) на новое основание.** |

**Перевод чисел из одной системы счисления в другую**

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 деления (умножения) на новое основание**

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

***Преобразование целых чисел****.*

Задачу представления числа *N*, заданного в системе *q*1, в системе счисления с основанием *q*2 можно рассматривать как задачу поиска коэффициентов полинома, представляющего собой расширенную запись числа *N* в системе счисления *q*2:



Отсюда вытекает правило формирования коэффициентов данного полинома или разрядов записи заданного числа *N* в системе счисления с основанием *q*2:

– необходимо разделить исходное число *Nq*1 на новое основание *q*2, при этом получив целое частное и остаток;

– полученный остаток снова необходимо разделить на *q*2, процесс деления продолжается до тех пор, пока частное будет не меньше нового основания *q*2*.* Если очередное сформированное частное будет меньше, чем *q*2, то процесс формирования записи заданного числа в новой системе с основанием *q*2 считается законченным, а в качестве искомых разрядов новой записи числа используются результаты выполненных

операций деления следующим образом:

– в качестве старшего разряда берется значение последнего частного, для остальных разрядов используются значения остатков в порядке, обратном порядку их получения.

*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) = 27и rest (54/2) = 0;

27/2 → int(27/2) = 13 и rest (27/2) =

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.

***Преобразование дробных чисел***

Задачу представления дробного числа *Mq*1, заданного в системе *q*1, в системе счисления с основанием *q*2, можно рассматривать как задачу поиска коэффициентов полинома, представляющего собой расширенную запись числа *M* в системе счисления *q*2:



Отсюда вытекает следующее правило формирования коэффициентов полинома, которые одновременно являются разрядами записи заданного числа *M* в системе счисления с основанием *q*2:

– определяется количество разрядов «*n*» в записи числа *Mq*2 в новой системе счисления;

– исходное число *Mq*1 умножается на *q*2, при этом будет получено смешанное число;

– дробная часть полученного произведения снова умножается на *q*2 и т. д.; процесс умножения повторяется *n* раз. В качестве искомых разрядов новой записи числа используются результаты выполненных операции деления следующим образом: в качестве первого старшего разряда искомой записи числа в новом основании берется

значение целой части первого произведения, в качестве второго старшего разряда искомой записи числа в новом основании берется значение целой части второго произведения и т. д.

|  |
| --- |
| **Вопрос 3. Перевод чисел из одной системы счисления в другую. Метод с использованием особого соотношения оснований исходной и искомой систем счисления.** |

**Перевод чисел из одной системы счисления в другую**

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 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:



**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 с новом основанием *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.

**Если имеет место *условие 1***, то запись заданного числа

*N* = *anаn*–1*аn–*2...*а*1*а*0

в системе с новом основанием *q*2 формируется следующим образом:

– исходная запись числа разбивается на группы по *m* разрядов, двигаясь от точки вправо и влево (недостающие разряды в крайних группах (слева и справа) дополняются нулями;

– каждой полученной группе ставится в соответствие цифра новой системы счисления;

– искомая запись заданного числа в новой системе счисления образуется из цифр, соответствующих группам, на которые была разбита исходная запись.

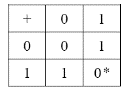
|  |
| --- |
| **Вопрос 4. Арифметические операции над двоичными числами. Операция сложения и вычитания в двоичной системе исчисления.** |

**Арифметические операции над двоичными числами**

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.

**Операция сложения и вычитания в двоичной системе исчисления**

Правила сложения в двоичной системе счисления



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.

При формировании поразрядной суммы и учете возникших переносов используется следующая классификация разрядов складываемых операндов:

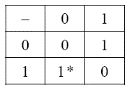
– разряд, генерирующий перенос (оба операнда в этом разряде имеют «1»);

– разряд, пропускающий перенос (операнды в этом разряде имеют разные значения);

– разряд, блокирующий распространение переноса (операнды в этом разряде имеют одинаковые значения).

Правила вычитания представлены ниже в форме таблицы.

***Правила вычитания в двоичной системе счисления***



Все возможные значения вычитаемого задаются во второй и третьей строках первой колонки; все возможные значения уменьшаемого – во второй и третьей колонках первой строки. На пересечении отмеченных значениями операндов строк и колонок располагается результат вычитания второго операнда из первого. В таблице знаком «*\**»

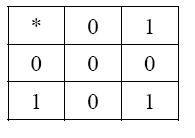
отмечен случай, когда в текущем разряде результата получена единица при займе из ближайшего старшего разряда.

Как видно из таблицы и приведенного примера, реализация операции вычитания не сложнее операции сложения. В ЭВМ никогда в перечне выполняемых операций арифметического устройства, 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.

|  |
| --- |
| **Вопрос 5. Операция умножения в двоичной системе исчисления.** |

**Операция умножения в двоичной системе исчисления**

Правило умножения в двоичной системе счисления задается в табличной форме.

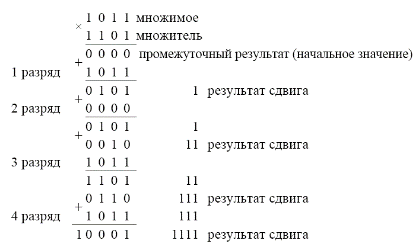


Все возможные значения множимого задаются во второй и третьей строках первой колонки; все возможные значения множителя – во второй и третьей колонках первой строки. На пересечении отмеченных значениями операндов строк и колонок располагается результат умножения первого операнда на второй. При умножении многоразрядных операндов, как правило (особенно это наглядно в десятичной системе счисления), используется метод, при котором формирование произведения выполняется за счет суммирования частичных произведений, которые оформляются посредством умножения множимого на отдельные разряды множителя с учетом веса соответствующего разряда множителя.

При реализации умножения рассматриваемым способом требуется использование *n*-разрядного регистра для хранения множимого, 2*n*-разрядного сумматора для подсчета промежуточных и конечного произведения и 2*n*-разрядного сдвигающего регистра для хранения множителя.

Реализация данного метода умножения требует использовать 2*n*-разрядный сумматор для последовательного, от такта к такту, формирования 2*n*-разрядного произведения и 2*n*-разрядный регистр для хранения и сдвига влево множимого.

Также есть метод умножения со сдвигом промежуточного результата. При использовании данного подхода умножение чисел 1101 и 1011 представляется в виде следующих действий:



В ЭВМ при выполнении различных операций, в том числе и операции умножения, разрядность операндов и результатов одинаковая.

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.

**Используя эти два классификационных признака, 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;

|  |
| --- |
| **Вопрос 6. IEEE754. Специальные числа. Зачем нулю знак.** |

**IEEE 754 — широко используемый стандарт, описывающий формат представления чисел с**[**плавающей точкой**](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B2%D0%B0%D1%8E%D1%89%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D1%8F%D1%82%D0%B0%D1%8F)**.**

Числа с плавающей точкой 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 точкой.

Тип данных с одинарной точностью (float). Числа с половинной, двойной и расширенной точностью обладают теми же особенностями, но имеют другой диапазон порядка и мантиссы. 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

  
-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 = 0,1562

**Специальные числа: 0, бесконечность и неопределенность**

В IEEE754 число «0» представляется значением с порядком, равным 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 - от not a number)**. Эти числа позволяет вернуть адекватное значение при переполнении. Бесконечности представлены как числа с порядком E=Emax+1 и нулевой мантиссой. Получить бесконечность можно при переполнении и при делении ненулевого числа на ноль. Бесконечность при делении разработчики определили исходя из существования пределов, когда делимое и делитель стремиться к какому-то числу. Соответственно, c/0==±∞ (например, 3/0=+∞, а -3/0=-∞), так как если делимое стремиться к константе, а делитель к нулю, предел равен бесконечности. При 0/0 предел не существует, поэтому результатом будет неопределенность. **Неопределенность или NaN– 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. Любая операция с NaN возвращает NaN. При желании в мантиссу можно записывать информацию, которую программа сможет интерпретировать. Стандартом это не оговорено и мантисса чаще всего игнорируется. 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 -0)**

В описанном представлении чисел с плавающей запятой существует два нуля, которые отличаются только знаком. 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.

|  |
| --- |
| **Вопрос 7. Деление двоичных чисел (общие правила).** |

**Деление двоичных чисел (общие правила)**

Деление в принципе является неточной операцией, поэтому при ее 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;

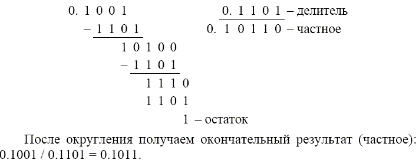
− упрощается процедура умножения найденной цифры частного на делитель.

*Пример*.

Найти частное от деления двоичных чисел 0.1001 на 0.1101.

*Решение*.

По умолчанию считается, что разрядность результата и операндов одинаковая, поэтому окончательный результат должен иметь в данном случае 4 разряда. Учитывая необходимость округления, найдем дополнительный пятый разряд, на основании которого выполним округление.



|  |
| --- |
| **Вопрос 8. Деление двоичных чисел с восстановлением остатка.** |

**Деление двоичных чисел с восстановлением остатка**

Алгоритм описывается следующим образом.

Пусть А - делимое, В - делитель, С - частное.

A=0,α1α2...αn; B=0,b1b2...bm, C=0,c1c2...cr.

На каждом шаге определяется остаток Аi=Аi–1 – В\*2-i, производится анализ, если остаток Аi>0, то старший разряд частного Ci=1, производится левый сдвиг и снос следующего разряда числа А, переход к определению следующего остатка. Если Ai<0, то Сi=0 и восстанавливается остаток Ai=Ai-1+B\*2-iна следующем шаге после сдвига определяется новый остаток и т.д.

|  |
| --- |
| **Вопрос 9. Деление двоичных чисел без восстановления остатка.** |

**Деление двоичных чисел без восстановления остатка**

Метод деления бинарных чисел без восстановления промежуточных остатков выполняется в последовательности:

* определить знак частного по формуле SgC= SgA+SgB;
* представить числа в дополнительном коде в машинном изображении, делимое берется в прямом коде с положительным знаком, а делитель берется в дополнительном коде с отрицательным знаком;

- присвоить сумматору значение См:=Амдоп , РгВ := Bмдoп; РгС:=0;

- устраняем дробную часть в делителе, перенося запятую вправо на n разрядов (по аналогии с десятичной системой счисления) и, чтобы дробь не изменилась, в делимом также переносим вправо запятую на n разрядов;

- начиная со старших разрядов, к делимому прибавляют делитель в дополнительном коде, что равносильно вычитанию из делимого делителя и анализируют знак промежуточного остатка:

1) если знак промежуточного остатка 00, то в регистр частного РгС записывается 1, начиная со старшего разряда. Остаток сдвигается на один разряд влево и сносится последующий разряд делимого не участвующий до этого в делении. После этого, промежуточный остаток подготовлен к последующему прибавлению делимого в дополнительном коде;

2) если знак промежуточного остатка 11, то в регистр частного Рг.С записывается 0, начиная со старшего разряда. Остаток сдвигается на один разряд влево и сносится последующий разряд делимого не участвующий до этого в делении. После этого, промежуточный остаток подготовлен к последующему прибавлению к нему делимого в прямом коде со знаком 00;

-действия предыдущего пункта повторяются до получения машинного нуля или заданной точности вычисления (количество разрядов дроби после запятой целой части числа). Запятая дроби устанавливается в частном после сноса последнего разряда целой части делимого.

- знак результату присваивается в соответствии с пунктом 1. Результат деления представлен в регистре частного в прямом коде.

|  |
| --- |
| **Вопрос 10. Двоично-десятичная арифметика. Сложение и вычитание двоично-десятичных чисел.** |

**Двоично-десятичная арифметика**

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.

**Сложение и вычитание двоично-десятичных чисел**

Сложение двоично-десятичных чисел

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 десятично-

го 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 таких тетрадах должен быть выработан перенос и, чтобы его выработать по правилам двоичной арифметики в тетраду нужно добавить шестерку. Для рассмотренного при-

мера такой тетрадой является и четвертая тетрада (отмечена \*\*).

Для получения двоично-десятичной разности *A* на основании результата вычитания операндов по правилам двоичной арифметики необходимо вычесть шестерку (0110) из тетрад, в которые пришел заем. Это обусловливается тем, что заем, сформированный по правилам двоичного вычитания, приносит в тетраду шестнадцать, а для десятичного

сложения заем должен был принести в тетраду десять, т. е. заем, сформированный по правилам двоичной арифметики, принес лишнюю шестерку. Для рассмотренного примера тетрадами, в которые пришел заем и в которых необходимо выполнить коррекцию (вычесть шестерку), являются вторая и четвертая тетрады (отмечены \*).

|  |
| --- |
| **Вопрос 11. Кодирование алгебраических чисел. Дополнительный и обратный коды двоичных чисел.** |

**Кодирование алгебраических чисел**

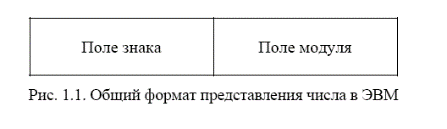
Для представления чисел со знаком используются специальные коды:

– *прямой код*;

*– дополнительный код*;

*– обратный код*.

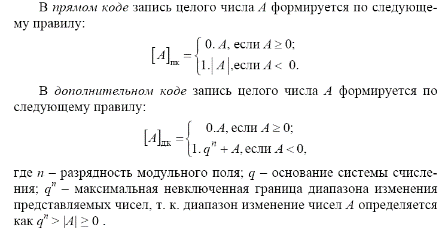
Во всех трех случаях используется следующий формат представления числа, содержащий два поля – поле знака и поле модуля

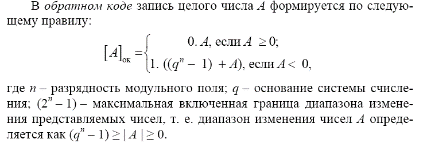


*Поле знака* представлено одним разрядом, в котором устанавливается 0, если число положительное, и 1, если число отрицательное.

*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.

**Дополнительный и обратный коды двоичных чисел**





***Правило формирования модуля обратного кода отрицательного двоичного числа.***

***Для формирования модульной части записи отрицательного числа в обратном коде достаточно в модульной части записи этого числа в прямом коде взять обратные значения всех двоичных разрядов, т. е. необходимо проинвертировать модуль прямого кода.* Переход от обратного кода отрицательного числа к представлению в прямом коде осуществляется по тому же правилу, т. е. необходимо проинвертировать модуль записи числа в обратном коде.**

***Правило формирования модуля дополнительного кода отрицательного числа.***

***Для формирования модульной части записи отрицательного числа в дополнительном коде достаточно в модульной части записи этого числа в прямом коде взять обратные значения всех двоичных разрядов, т. е. необходимо проинвертировать модуль прямого кода и к полученному коду прибавить 1 в младший разряд. Переход от дополнительного кода отрицательного числа к прямому осуществляется по тому же правилу, т. е. необходимо проинвертировать модуль записи числа в дополнительном коде, и к полученному коду прибавить 1 в младший разряд.***

**Как правило, информация в памяти хранится в прямом коде, а при выполнении операций применяется или обратный, или дополнительный код.**

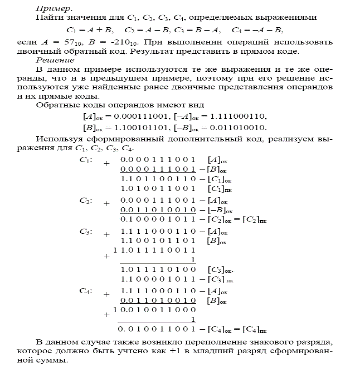
|  |
| --- |
| **Вопрос 12. Операции с двоичными числами в дополнительном и обратном кодах.** |

**Операции с двоичными числами в дополнительном и обратном кодах**

При сложении чисел, представленных в **дополнительном** коде, выполняется сложение разрядов, представляющих запись операндов, по правилам двоичной арифметики по всей длине записи чисел, не обращая внимание на границу, разделяющую знаковое и модульные поля. Переполнение знакового поля, т. е. перенос, возникший из крайнего левого разряда, игнорируется. В результате такого сложения будет получен дополнительный код суммы заданных операндов.

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.

При использовании **дополнительного** или **обратного** кода **операция вычитания** заменяется операцией сложения с изменением знака второго операнда.



|  |
| --- |
| **Вопрос 13. Модифицированные коды** |

**Модифицированные коды**

При расчете разрядности *n* модульного поля весьма трудно бывает учесть диапазон значений результатов, особенно когда последовательность операции, представленных в подлежащих реализации выражениях, достаточно сложна и тогда могут возникнуть ситуации переполнения.

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.*

**Если в результате сложения чисел в модифицированном коде полученный результат имеет в поле знака одинаковые значения в обоих разрядах (00 или 11), то переполнения нет, если же разряды знакового поля имеют неодинаковые значения (10 или 01), то имеет место переполнение.**

|  |
| --- |
| **Вопрос 14. Логические операции с двоичными кодами: логическое суммирование, логическое умножение, логическое отрицание, суммирование по модулю два, логические сдвиги.** |

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

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

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

*– отрицание* (обозначения – НЕТ, NOT, );

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

*– операции сдвига*.

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.

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.

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.

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.

Операции **логического *сдвига***в свою очередь, подразделяются на:

**сдвиг *влево***выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем младшем разряде (освобождающийся самый правый т. е. самый младший, разряд заполняется 0, а «выталкиваемый» разряд пропадает). Например, код 11001110 после сдвига влево будет иметь вид 10011100.

**сдвиг *вправо***выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем старшем разряде (в освобождающийся самый левый, т. е. самый старший, разряд заполняется 0, «выталкиваемый» разряд пропадает). Например, код 11001110 после сдвига влево будет иметь вид 01100111.

***циклический сдвиг влево***выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем младшем разряде (в освобождающийся самый правый, т. е. самый младший, разряд заносится значение старшего, т. е. самого левого разряда исходного кода).

Например, код 11001110 после сдвига влево будет иметь вид 10011101.

***циклический сдвиг вправо***выполняется за счет установки в разряд значения, соответствующего исходному значению в ближайшем старшем разряде (в освобождающийся самый левый т. е. самый старший, разряд заполняется значение в самом младшем разряде исходного кода).

Например, код 11001110 после сдвига влево будет иметь вид 01100111.

|  |
| --- |
| **Вопрос 15. 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**

**Арифметический сдвиг влево либо вправо положительных двоичных чисел выполняются независимо от используемого кода (прямого, обратного, дополнительного).** **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 этом при любом сдвиге вправо предусматриваются средства для округления после завершения нужного количества сдвигов и средства обнаружения обнуления сдвигаемой величины после очередного сдвига.

Реализация иллюстрируются следующими примерами:

*Примеры*.

**1.** 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.

2. Найти результат арифметического сдвига вправо на два разряда двоичного прямого кода числа [*А*]пк = 00.00000110.

*Решение.*

первый сдвиг: 00. 00000110 → 00. 00000011;

второй сдвиг: 00. 00000011 → 00. 00000001;

После выполнения заданного количества сдвигов выполняется округление на основании последнего «вытолкнутого» разряда; в данном случае последний «вытолкнутый» разряд равен 1, поэтому конечный результат выполнения заданного сдвига будет равен 00.00000010.

**3**. Найти результат арифметического сдвига вправо на четыре разряда двоичного прямого кода числа [*А*]пк = 00.00000110.

*Решение*.

первый сдвиг: 00.00000110 → 00.00000011;

второй сдвиг: 00.00000011 → 00.00000001;

третий сдвиг: 00.00000001 → 00.00000000.

После выполнения третьего сдвига будет выработан сигнал о получении нулевого результата. Оставшиеся сдвиги могут не выполняться.

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

**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 осуществляться контроль за обнулением результата сдвига (появление единичных значений во всех разрядах) и округление результата после выполнения заданного количества сдвигов.

|  |
| --- |
| **Вопрос 16. Арифметические сдвиги двоичных чисел, представленных в дополнительном коде. Сдвиг отрицательных чисел с переполнением.** |

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

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 (5010), представ-

ленного в дополнительном коде, дает в результате 11.10011100 (10010).

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 осуществляться контроль за обнулением результата сдвига (появление единичных значений во всех разрядах).

**Сдвиг отрицательных чисел с переполнением.**

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.

|  |
| --- |
| **Вопрос 17. Представление чисел с фиксированной точкой. Арифметические операции над числами, представленными с фиксированной точкой.** |

**Представление чисел с фиксированной точкой**

Числовая информация представляется в машине в форме с фиксированной или с плавающей точкой. При представлении с фиксированной точкой положение последней в записи числа фиксировано. Как правило, при использовании фиксированной точки числа представляются в виде целого числа или правильной дроби Точка в записи числа не отображается, а так как она находится всегда в одном месте, то указание на её положение в записи числа отсутствует. При n-разрядном представлении модульной части формат с фиксированной точкой обеспечивает диапазон изменения абсолютного значения числа А, для которого выполняется неравенство



Одним из важнейших параметров представления чисел является ошибка представления. Ошибка представления может быть абсолютной или относительной. Ошибка при

фиксированной точке может достигать сравнительно большого значения – 50%.

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

К числу основных арифметических операций, непосредственно реализуемых в ЭВМ, относятся операции **сложения, умножения, деления**. Остальные операции (например, такие, как возведение в степень, извлечение квадратного корня) реализуются программным способом.

Выполнение длинных операций, таких, как умножение и деление, реализуется в два этапа:

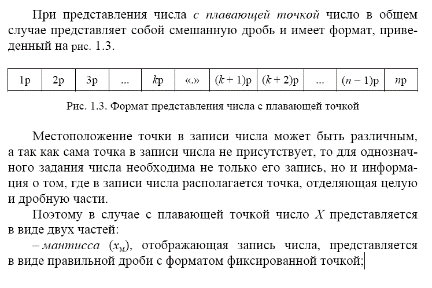
- на первом этапе формируется знак искомого результата,

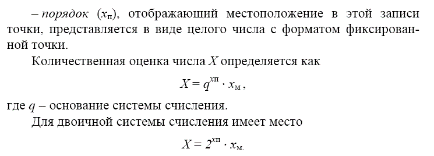
- на втором этапе, используя абсолютные значения операндов, ищем результат присваивание результату знак.

Операнды, как правило, представлены в прямом коде, и знак результата, не зависимо от того, частное это или произведение, ищется за счет сложения по модулю 2 знаковых разрядов операндов. В результате этого знак результата положителен, если операнды имеют одинаковые знаки, или отрицательный, если операнды имеют разные знаки.

|  |
| --- |
| **Вопрос 18. Представление чисел с плавающей точкой. Сложение чисел, представленных в формате с плавающей точкой** |

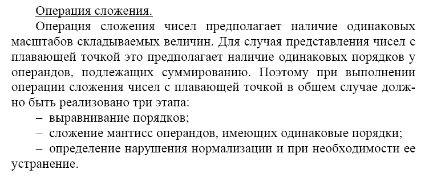
**Представление чисел с плавающей точкой**





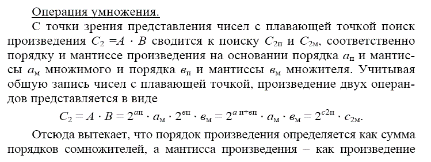
**Относительная ошибка при представлении чисел в форме с плавающей точкой существенно меньше, чем в случае с фиксированной точкой.**

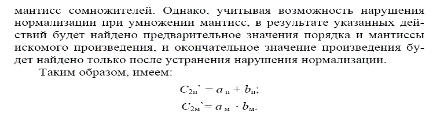
**Сложение чисел, представленных в формате с плавающей точкой**



|  |
| --- |
| **Вопрос 19. Умножение чисел, представленных в формате с плавающей точкой. Деление чисел, представленных в формате с плавающей точкой.** |

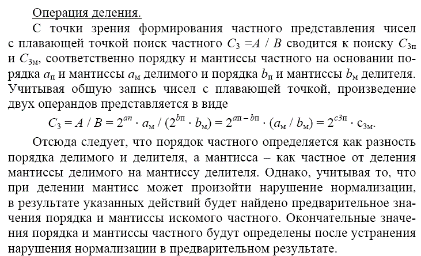
**Умножение чисел, представленных в формате с плавающей точкой**





плюс устраняется нарушение нормализации мантиссы произведения (если нарушение имеет место) соответствующей корректировкой предварительного значения порядка и мантиссы искомого произведения.

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



|  |
| --- |
| **Вопрос 20. Неосновные арифметические операции. Вычисление квадратного корня** |

**Неосновные арифметические операции**

К числу основных арифметических операций, непосредственно реализуемых в ЭВМ, относятся операции сложения, умножения, деления. Остальные операции (например, такие, как возведение в степень, извлечение квадратного корня) реализуются программным способом.

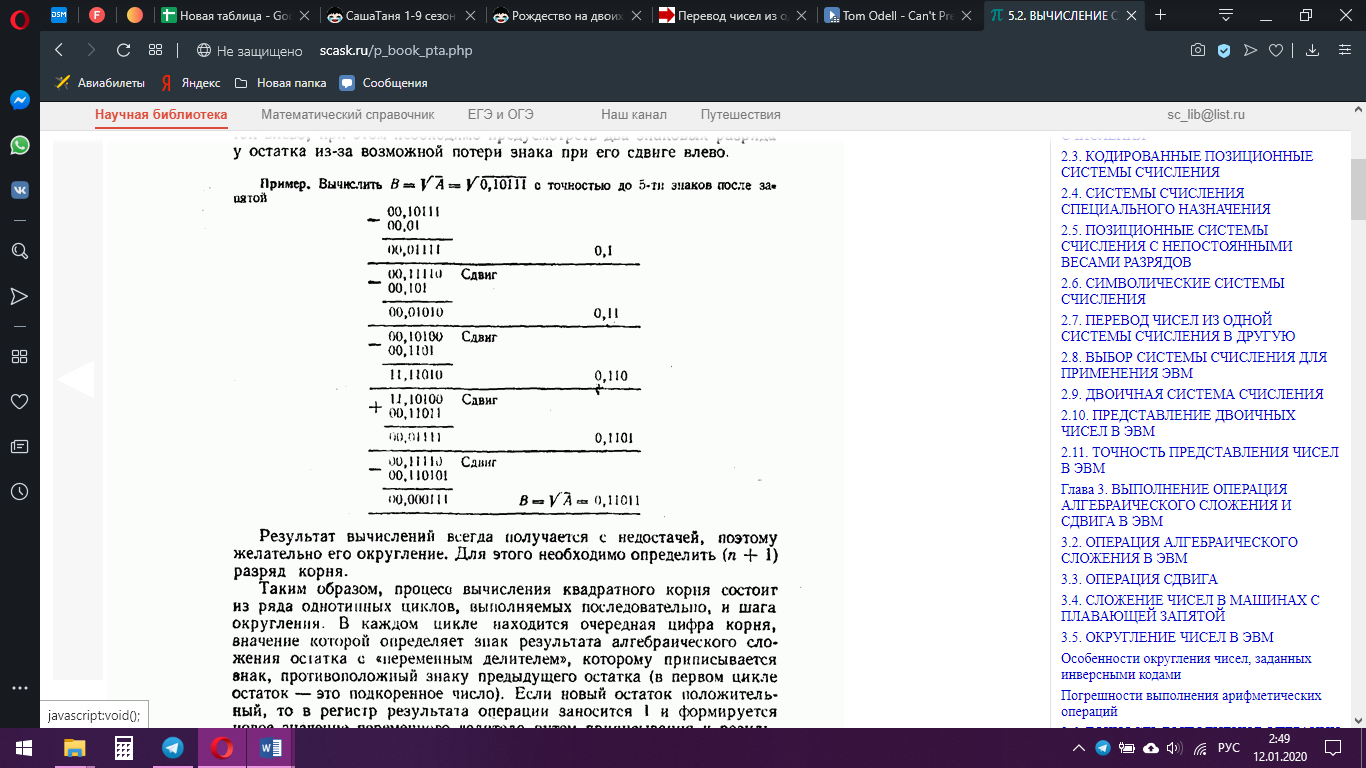
**Вычисление квадратного корня**

Имеются 2 пути вычисления квадратного корня:

1. Связан с разработкой микропрограммы извлечения квадратного корня с использованием набора простых арифметических операций. При этом микропрограмма реализует один из известных итерационных методов извлечения квадратного корня с помощью базовой аппаратуры.

Например, формулой Ньютона: Bi+1=0.5(Bi+A/Bi), где Bi+1 есть (i+1)-e приближение B=sqrt(A), а i=0,1,2…

2. Наиболее простой алгоритм сводится к подбору цифр в результате разряд за разрядом, начиная со старшего, т.е с 2-1. При этом вычисление i=q цифры D происходит следующим образом. После получения (i-1)-й цифры bi-1 в i-й разряд В для пробы помещается 1. Вычисляется разность (A-Bi2)=Rt. Если Rt>0, то Bi есть число, у которого цифры всех i разрядов совпадают с цифрами искомого результата B. Если Ri<0, то в i-м разряде bi нужно поставить 0 и переходить к вычислению (I +1)-го разряда.Т.к вычисление этого разряда снова начинается с подстановки пробной 1, то в случае (A-Bi2)=R<0 можно вместо «стирания» 1 в i-м разряде вычесть 1 из (i+1)-го разряда.



|  |
| --- |
| **Вопрос 21. Методы вычисления элементарных функций.** |

**Методы вычисления элементарных функций**

Методы вычисления элементарных функций:

* Разложение в ряд Тейлора (степенные полиномы)

Полиномы вычисляются по схеме Горнера. При этом требуется выполнить m операций умножения и m операций сложений (m- степень полинома). Используется редко, т.к. ЭВМ долго считает

* Аппроксимация с помощью различных полиномов

Используется наиболее часто. 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.

* 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.

* Рациональные приближения ЭФ
* Использование цепных дробей
* Итерационные (рекуррентные)

|  |
| --- |
| **Вопрос 22. Денормализованные числа. Подводные камни в арифметике с плавающей запятой.** |

**Денормализованные числа**

**Денормализованное** число - вид чисел с плавающей точкой, определенный в стандарте IEEE 754. При записи в форматах float, double, long double их экспонента будет записана как 0. Для получения их значения не требуется использование неявной единицы; мантисса просто умножается на наименьшую для данного формата экспоненту.

**Подводные камни в арифметике с плавающей запятой**

**Округление**  
Особенности округления в IEEE754:

Математически показано, что, если 0,5 округлять до 1, существует набор операций, при которых ошибка округления будет возрастать. 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. 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)».

**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

**Числовые константы**

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». Абсолютная погрешность в отдельном случае может и не высока, но если использовать такую константу в цикле, можем получить накопленную погрешность.

**Выбор минимума из 2 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

**Сравнение чисел**

Лучший, но все равно ошибочный способ, это сравнивать разницу с допустимой абсолютной погрешностью. 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. Выбрать правильную процедуру сравнения сложно.

|  |
| --- |
| **Вопрос 23. Погрешности обусловленные форматом с плавающей точкой** |

**Погрешности, обусловленные форматом с плавающей точкой**

В случае с плавающей точкой число Х представляется в виде двух частей: мантисса и порядок. Количественная оценка числа Х определяется как Х = qxп × хм, где q – основание системы счисления. Для плавающей точки максимальные значения абсолютной и относительной ошибок определяются следующим образом:

**Максимальная абсолютная погрешность представления чисел:**

Dmax = 2-(s+1) × 2p; **Абсолютной погрешностью называется разность между этим числом и его точным значением (из большего числа вычитается меньшее)\*. Δ = x2 - x1.**

**Максимальная относительная погрешность:**

dmax = D max / Аmin = 2-(s+1) × 2p / (хм min × 2p)= 2­(s+1) × 2p / ( 2-1 × 2p) = 2­(s+1) / (2-1) = =2-s. **Относительной погрешностью приближенного числа называется отношение абсолютной погрешности приближенного числа к самому этому числу. δ = Δ/a.**

**Относительная ошибка при представлении чисел в форме с плавающей точкой существенно меньше, чем в случае с фиксированной точкой. Это, а также больший диапазон изменения представляемых чисел, является основным преимуществом представления чисел с плавающей точкой.**

**24.Основн понятия алгебры логики. Способы задания логич функц.**

**Алгебра логики** используется при анализе и синтезе схем ЭВМ.

Причины:

1) соответствие представления переменных и функций алгебры логики.

2)двоичным представлением информации и характер работы отдельных компонентов вычислительной техники.

**Логич. переменная** -это переменная, которая может принимать одно из двух значений.

**Логич. константа** -это постоянная величина, значением которой может быть истинно или ложно.

**Логич. функция** -это функция, которая может принимать одно из двух значений: истинно или ложно в зависимости от текущих значений ее аргументов, в качестве которых используются логические переменные.

Зависимость логич функции от переменных может задаваться по–разному:

1)словесным описанием(используется в случае сравнительно несложной логической функции).

2)таблицей истинности (она включает все наборы для заданного количества переменных, определяющих значение логической функции, с указанием значений, которые принимает функция для каждого набора, Максимальное количество полностью определенных функций от «n» переменных определяется как M = (2^2)^n)

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

**25.Понятие о принципе двойственности. Суперпозиция логических функц.**

**Двойственные функции**- если одна функция получается из другой заменой каждой операции конъюнкции на операцию дизъюнкции и наоборот.

**Принцип двойственности** формулируется так: если функции F1 иF2 равносильны, то равносильны и двойственные им функции F1\* и F2\*

**Суперпозицией функц-** функц полученная из некоторых функц путем применения принципа суперпозиции.

Все логические функции двух переменных, а также конъюнкции и дизъюнкции n переменных называются **элементарн логич функц.**

Они позволяют строить любые новые функции алгебры логики, являющиеся суперпозициями элементарных функций

**26. Нормальная и совершенные нормальные логических функций**

**Дизъюнкция** любого числа элементарных конъюнкций называется **дизъюнктивной нормальной формой** (ДНФ) Например: а+б\*с+(не а)б\*с+а\*(не б \* не с**).**

**Конъюнкция** любого числа элементарных дизъюнкций называется **конъюнктивной нормальной формой** (КНФ). Например: ф(ф+б)(не б+с)(не а+ б+ не с)

Для каждой функции может существовать несколько дизъюнктивных и конъюнктивных нормальных форм (являющиеся равносильными друг другу)

**Совершенно нормальная функц-**1 вид дизъюнктивной норм форм и конъюнктивной норм форм, в которых функция может быть записана только единственным образом.

**СДНФ** – дизъюнкция конституентов единицы тех наборов значений переменных, где данная функция равна единице.

**СКНФ** – конъюнкция конституентов нуля тех наборов, где данная функция равна нулю.

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

**Задача** минимизации булевых функций: найти аналитическое выражение заданной булевой функции в форме, содержащей минимально возможное число букв.

**Элементарной конъюнкцией** – конъюнкц конечного числа различных между собой булевых переменных, каждая из которых может иметь или не иметь отрицания.

**ДНФ** называется дизъюнкция элементарных конъюнкций .Минимальной ДНФ булевой функции называется ДНФ, содержащая минимальное число букв.

Булева функция g(x1,x2…xn)называется **импликантной** булевой функции f(x1,x2…xn), если для любого набора переменных, на некотором g=1, справедливо f=1.Импликанта g булевой функции f, являющаяся элементарной конъюнкцией, называется простой, если никакая часть импликанты g не является импликантой функции f.Сокращенная ДНФ булевой функции называется тупиковой, если в ней отсутствуют лишние простые импликанты

**Наиб. известные методы минимиз:**Методом Квайна ,использование диаграммы Вейча,Не полностью определенных функций,Конъюнктивных нормальных форм,Метод кубического задания функций алгебры логики,Методом Квайна-Мак-Классики,Методом Нельсона,С использованием алгоритма извлечения,ФАЛ методом преобразования логических функций

**Минимизация системы логических функций происходит по следующему алгоритму:**

1.Построить полное множество А элементарных конъюнкций минимизируемой системы функций, считая что вначале каждая из функций системы представлена в СДНФ. Каждой единице констнтуента множества присвоить знак, содержащий номера функций системы, в которые входит рассматриваемая констнтуета.

2.Произвести минимизацию СДНФ функции.

3.Построить импланкантную матрицу функции, аналогичную матрице Квайна с той разницей, что для каждой констнтуенты единицы выделяются столько столбцов, сколько различных номеров функций содержит ее знак

**Минимизация частично определённых функций происходит по алгоритму:**

1.Нахождение любым способом сокращенную ДНФ/КНФ функции, получающейся до определением единицами исходной функции f на всех неопределенных наборах.

2.Выбрать минимальную ДНФ/КНФ по импликантной матрице, где в столбцах записаны лишь те конституанты единицы функции f, которые соответствуют полностью определенным единичным наборам

**28.Минимизация логических выражений методом Квайна.**

В качестве исходной формы представления логического выражения используется СДНФ. Метод Квайна выполняется в два этапа.

1)цель получить тупиковую форму, 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*.**

**2)**цель устранить из тупиковой формы всех избыточных простых импликант, что дает в результате минимальное логическое выражение.

**29.Минимизац логич выраж с использованием Карт Карно(диаграммамиВейча).  
Минимизац** - предполаг использование специальных форм – диаграмм Вейча (или карт *Карно*). Карта Карно для «*n*» логических переменных представляет собой множество квадратов (клеток) .Записываемая функц должна быть представлена в СДНФ. Запись функции в карту осуществляется за счет установки «1» в клетки карты. Для выполнения минимизации представленной в карте Карно функции необходимо выполнить два этапа:

1)охватить множество клеток карты Карно контурами;

**2)**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:

1.контур должен иметь прямоугольную форму; должно быть минимальное количество; контурами необходимо охватить все клетки с единичными значениями;2.в контур может входить количество клеток, равное целой степени числа «2», могут входить клетки, являющиеся логическими соседями, необходимо включить максимальное количество клеток с учетом вышеприведенных требований;количество должно быть равно 2DR, где DR –разность ранга (дельта ранга) конституент единицы заданной функции и ранга конъюнкции, соответствующей контуру.  
**Логическими соседями** являются две клетки, наборы которых отличаются только одной переменной – в одном эта переменная должна иметь прямое, в другом – обратное значение**.З**апись мин выражения по заданной функции имеет вид дизъюнкции простых конъюнкций, соответствующих контурам на карте, и формируется образом:конъюнкция, соответствующая контуру, должна включать только те переменные, которые имеют постоянное значение во всех клетках, охваченных рассматриваемым контуром.

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

Саму функцию надо привести к одному из указанных базисов путём использования правил Моргана и различных законов чтобы все действия были или только конъюкцией или только дизъюнкцией. Далее надо будет построить схему с шиной которая будет выполнять полученную функцию.(чекать 31 вопрос)

**31. Законы и правила алгебры Буля**

При работе с булевыми логич выраж используются след законы, правила и операции.

***Переместительный* *закон.*** *(*Закон справедлив как для конъюнкции и для дизъюнкции, справедлив для любого количества операндов).от перемены мест логических слагаемых сумма не меняется,от перемены мест логических сомножителей их произведение не меняется.

***Сочетательный закон****. (*Справедлив как для конъюнкции, так и для дизъюнкции.)при логическом сложении отдельные слагаемые можно заменить их суммой,при логическом умножении отдельные логические сомножители можно заменить их произведением

***Распределительный* *закон*.** (х1 + х2) х3 = х1х3 + х2х3; (х1 + х2) (х1 + х3) = х1 + х2х3

***Правило де Морган:.***отрицание суммы равно произведению отрицаний,отрицание произведения равно сумме отрицаний.

***Операция склеивания****:*для конъюнкций, где А – переменная или любое логическое выражение.операция склеивания для дизъюнкций

***Операции с отрицаниями:***двойное отрицание равносильно отсутствию отрицания

***Операции с константами.Операции с одинаковыми операндами.***

**32.Параллелизм применения  
Пространственный** **параллелизм** предполагает наличие нескольких устройств, на которых одновременно могут выполняться независимые команды. Команды считаются независимыми, если они не имеют между собой зависимостей по данным, по управлению или по ресурсам.

Использование параллелизма ограничивается ***взаимозависимостями*** реальных задач. Если текущая задача зависит от результатов предыдущей задачи, а не только от своих предыдущих шагов, то выполнение задачи не может быть начато до завершения предыдущей задачи. **Параллелизм** – один из самых важных методов проектирования высокопроизводительных цифровых систем.

**33.Устройства ЭВМ. Состав АЛУ.**

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

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.

Для передачи информации между отдельными узлами используются шины Ш1 – Ш3. Шина Ш3 обеспечивает также связь с запоминающими устройствам ЗУ (ЭВМ).

**34.Типы памяти**

**Память ЭВМ** -совокупность устройств, служащих для запоминания, хранения и выдачи информации. Отдельные устройства, входящие в эту совокупность, называются запоминающими устройствами (ЗУ) того или иного типа.Основным параметры характер запоминающие устройства, относятся емкость (это максимальное количество данных, которое в ней может храниться.) и быстродействие(Быстродейств памяти определяется продолжительностью операции обращения, то есть временем, затрачиваемым на поиск нужной информации в памяти и на ее считывание, или временем на поиск места в памяти, предназначенного для хранения данной информации)

**Регистровая память** – Регистры CPU программно доступны и хранят информацию, наиболее часто используемую при выполнении программы.

**Оперативная память** – устройство, которое служит для хранения информации ,используется в ходе выполнения программы в процессоре

**Кэш-память** - более быстродействующая статическая оперативная память со специальным механизмом записи и считывания информации и предназначена для хранения информации.Как правило, часть кэш-памяти располагается непосредственно на кристалле микропроцессора (внутренний кэш), а часть – вне его (внешняя кэш-памятьДля обращения к ней используются аппаратные средства процессора и компьютера.

**Внешняя память** - это **память**, реализованная в виде **внешних**, относительно материнской платы, устройств с разными принципами хранения информации и типами носителя, предназначенных для долговременного хранения информации.

**35.Код Грея**-[двоичный код](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), в котором две «соседние» кодовые комбинации различаются только цифрой в одном двоичном разряде. Иными словами, [расстояние Хэмминга](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_%D0%A5%D1%8D%D0%BC%D0%BC%D0%B8%D0%BD%D0%B3%D0%B0) между соседними кодовыми комбинациями равно 1. Используется для упрощения выявления и [исправления ошибок](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BD%D0%B0%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B8_%D0%B8%D1%81%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA) в системах связи, а также в формировании сигналов обратной связи в системах управления.Используется в передаче меняющихся цифровых сигналов в отсутствие [тактового сигнала синхронизации](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%BA%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB). Например, код (обычный двоичный) перескакивает 3→4, или 0112 → 1002. Если из-за несовершенства считывателя мы прочитаем первый бит от 011, а остальные два — от 100, мы получим 0002=0 — число, далёкое от реальных значений. В коде Грея никаких посторонних значений не будет: перескок будет в одном разряде, 010G → 110G, и мы считаем либо старое 010G=3, либо новое 110G=4.

**36. Обратная польская запись.**

*Обратная польская запись(ОПН)*— форма записи математич и логич выраж, в котор операнды располож перед знаками операций.

Пример: *(1+2)\*4+3 в ОПН: 1 2 + 4 × 3 +* ***или*** *(8+2\*5)/(1+3\*2-4).В ОПН:* *825\*+132\*+4-/*

*Приоритеты знаков: \* / + - (* (от больш к меньш)

**37. АЦП и ЦАП.**

*Цифро-аналоговый преобразователь*— устройство для преобразов цифрового кода в аналоговый сигнал. Цифро-аналоговые преобразователи явл интерфейсом между дискретным цифровым миром и аналоговыми сигналами.

*Аналого-цифровой преобразователь* — устройство, преобразующее входной аналоговый сигнал в дискретный код (цифровой сигнал).

АЦП преобразует напряжение в двоичный цифровой код, а ЦАП наоборот.

**38. Корректирующие коды. Код Хэ́мминга. Область применения.**

*Корректирующие коды* — это коды, которые могут обнаружить ошибки, возникшие при передаче данных. Для того чтобы «код обладал коррект спосбнстями, в кодовой последоват должны содержаться дополнительные(избыточные) символы, предназначен для корректир ошибок. Чем больше избыточность кода, тем выше его корректирующая способность. К основным характеристикам кор.кодов относятся: ● число разрешённых и запрещённых кодовых комбинаций; ●избыточность кода(Величина = k (число информационных символов)/n (число информационных символов); ● минимальное кодовое расстояние (dmin) (минимальное число искаженных символов, необходимое для перехода одной разрешенной комбинации в другую); ●число обнаруживаемых или исправляемых ошибок; ●Корректирующие возможности кодов.

*Код Хэмминга* —первый самоконтролирующийся и самокорректирующийся код (Коды, в которых возможно автоматическое исправление ошибок). Построен применительно к двоичной системе счисления. Позволяет исправлять одиночную ошибку и находить двойную. Построение кодов Хэмминга основано на принципе проверки на четность числа единичных символов: к последовательности добавляется такой элемент, чтобы число единичных символов в получившейся последовательности было четным. Характеристикой самок.кода явл: ●Числ разрешенных и запрещенных комбинаций. ●Избыточность кода. ●Минимальное кодовое расстояние ●Числ обнаруживаемых и исправляемых ошибок. ●Корректирующие возможности кодов.

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

**39. Языки описания аппаратуры. ПЛИС (FPGA) модули.**

*Язык описания аппаратуры* — специализиров компьютерный язык, используемый для описания структуры и поведения эл схем, чаще цифровых логических схем. Языки описания аппаратуры внешне похожи на такие языки программир, как Си или Паскаль Важнейшим отличием является явное включение концепции времени в языки описания аппаратуры.

*FPGA* — программируемая логическая интегральная схема (ПЛИС) Для описания и верификации проектов на основе FPGA используют языки высокого уровня, такие как VHDL и Verilog. Возможна интеграция проектов на языках С, C++ сама микросхема ПЛИС состоит из: ● конфигурируемых логических блоков, реализующих требуемую логическую функцию; ● программируемых эл связей между конфигурируемыми логич блоками; ● программируемых блоков ввода/вывода, обеспечивающих связь внешнего вывода микросхемы с внутренней логикой.

**40. Сумматор. Многоразрядный сумматор.**

Сумматор-устр-во котор обеспечивает сложение одноименных разрядов операндов с учетом переноса, поступающего от ближайшего младшего разряда. Сумматор вырабат знач соответствующего разряда суммы (S) и перенос (P), который должен быть учтен в соседнем старшем разряде. Синтез схемы, реализующей функции одноразрядного сумматора, можно выполнить на основании таблицы истинности.

Многоразрядный двоичный сумматор строится на основе одноразрядных сумматоров с введением соответств связей между разрядами. Сумматор обладает малым быстродействием из-за последовательн 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.

**41. Полная система логических функций.**

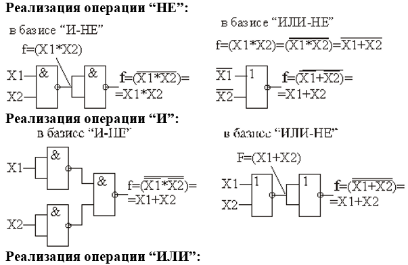
Функционально полная система логич функций представл собой набор логич функций, с помощью которых можно записать любую сложную функцию. В этом случае говорят, что этот набор образует базис. Функционально полными являются 3 базиса:

1) "И-ИЛИ-НЕ" (базис конъюнкции, дизъюнкции, инверсии)

2) "И-НЕ" (базис Шеффера)

3) "ИЛИ-НЕ" (базис Пирса или функция Вебба).









**42. Искусство управления сложностью. Цифровая абстракция.**

1)Важен систематический подход к управлению сложностью многоуровневой системы. ***Абстракция*** – принцип управления сложностью системы, подразумевающий исключение из рассмотрения тех элементов, которые в данном конкретном случае несущественны для понимания работы этой системы.

***Три базовых принципа*** для управления сложностью системы*: иерархичность, модульность конструкции и регулярность.* Эти принципы применительны как к ПО, так и к аппаратной части компьютерных систем.

*Иерархичность* – принцип иерархичности предполагает разделение системы на отдельные модули, а затем последующее разделение каждого такого модуля на фрагменты до уровня, позволяющего легко понять поведение каждого конкретного фрагмента.

*Модульность* – принцип модульности требует, чтобы каждый модуль в системе имел четко определенную функциональность и набор интерфейсов и мог быть легко и без непредвиденных побочных эффектов соединен с другими модулями системы.

*Регулярность* – принцип регулярности требует соблюдения единообразия при проектировании отдельных модулей системы.

2) Джордж Буль разработал систему логики, использующую двоичные переменные, и эту систему сегодня называют – Булева логика. Булевы переменные могут принимать значения ИСТИНА (TRUE)(1) или ЛОЖЬ (FALSE)(0).

***Преимущества цифровой абстракции*** заключ в том, что разработчик цифровой системы может сосредоточиться исключит на единицах и нулях, игнорируя, каким образом булевы переменные представлены на физическом уровне.

**43. Логические элементы. Таблицы истинности. Обозначения элементов в разных представления.**

Таблица истинности — таблица, описывающая логическую функцию.





**44. За пределами цифровой абстракции. Напряжение питания. Логические уровни. Допускаемые уровни шумов.**

1)Цифров сист оперирует дискретными переменными(0, 1). Для представл этих переменных использ непрерывные физич величины, такие как напряжение в эл цепи.

2) Самое высокое напряжение в системе поступает от блока питания и обознач VDD.

3) Отображение непрерывно-меняющейся переменной на различные значения дискретной двоичной переменной выполняется путем определения логических уровней. Первый логический элемент в рассматриваемой схеме называется источник, а второй – приемник. 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. VOH и VOL называются соответственно высоким и низким логическими уровнями выхода, а VIH и VIL называются соответственно высоким и низким логическими уровнями входа.

4) *Допускаемый уровень шумов* – это то max кол-во шума, присутствие которого в выходном сигнале источника не мешает приемнику корректно интерпретировать знач получен сигнала. Знач нижнего допускаемого уровня шумов и верхнего допускаемого уровня шумов определяются следующим образом: *NML = VIL – VOL и NMH = VOL – VIH.*

**45. Передаточная характеристика. Статическая дисциплина.**

*Передаточная характеристика* какого-либо логич вентиля описывает напряж на выходе этого элемента как ф-ию напряжения на его входе, когда входной сигнал изменяется настолько медленно, что выходной сигнал успевает изменяться вслед за ним. Такая характеристика назыв передаточной, тк описывает взаимосвязь между входным и выходным напряжением.

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

**46. Биполярные и КМОП транзисторы. Полупроводники. Конденсаторы. n-МОП и p-МОП транзисторы.**

1)В технологии КМОП используется «полевой транзистор» управляется он электрическим полем затвора, в то время как биполярный транзистор управляется током базы.

2) Полупроводник — материал, который по своей удельной проводимости занимает промежуточное место между проводниками и диэлектриками. МОП-транзисторы изготавл из кремния. Кремний – плохой проводник, но его проводимость улучшается, если добавить в него небольшое кол-во атомов другого вещества. Если в качестве примеси используется элемент, например, мышьяк, то в каждом атоме примеси окажется дополнительный электрон, не участвующий в образовании ковалентных связей. Если же в качестве примеси используется бор, то в каждом из атомов примеси будет не хватать одного электрона. Отсутствующий электрон называют дыркой.

3)Конденсатор состоит из двух проводников, отделенных друг от друга изолятором. Если к одному из проводников приложить напряжение, то через некоторое время этот проводник накопит эл заряд, а другой проводник накопит противоположный электрич заряд −Q.

4)Полевой МОП-транзистор ведет себя как переключатель, управляемый приложенным к нему напряжением. Он представляет собой «сэндвич» из нескольких слоёв проводящих и изолирующих материалов. Сущ два вида полевых МОП-транзисторов: n-МОП и p-МОП. В транзисторах n-типа, области, где расположены полупроводниковые примеси n-типа – в свою очередь называемые истоком и стоком – находятся рядом с затвором, причем вся эта структура размещается на подложке p-типа. В транзисторах же p-МОП и исток, и сток – это области p-типа, размещенные на подложке n-типа.

**47. Логический вентиль НЕ и другие на КМОП-транзисторах. Псевдо n-МОП-Логика. Потребляемая мощность.**

1.1)Схема на Рис. 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. n-МОП-транзистор N1 включен между землей GND и выходным контактом Y. В свою очередь, p-МОП-транзистор P1 включен между напряжением питания VDD и выходным контактом Y. Напряже-ние на входном контакте А управляет переходами обоих транзисторов. Если напряжение на А равно 0, то транзистор N1 выключен, транзистор P1 включен. При этом, напряжение на контакте Y равно напряжению питания VDD, а не земли, что соответствует логич единице(1)ю. . Если же напряжение на контакте А равно логической единице, то транзистор N1 включен, а транзи-стор P1 выключен, и напряжение на контакте Y равно напряжению земли, что соответствует логиче-скому нулю. В этом случае говорят, что Y «подтянут» к нулю. Включенный транзистор N1 хорошо передает логический ноль, то есть напряжение на контакте Y очень близко к GND.

1.2) Другие логические вентили на КМОП-транзисторах. На Рис. показана схема для построения с помощью МОП-транзисторов логического элемента И-НЕ с двумя входными контактами. На Рис. n-МОП-транзисторы N1 и N2 соединены последовательно. Причем, чтобы замкнуть выходной контакт на землю GND – то есть понизить логический уровень, оба этих транзистора должны быть включены. В то время как p-МОП-транзисторы P1 и P2 соединены параллельно, и только один из них должен быть включен, чтобы соединить выходной контакт с напряжением питания VDD – то есть повысить логический уровень.

2) Построенный по технологии КМОП логический вентиль ИЛИ-НЕ, число входных контактов = N, использ N параллельно включенных n-МОП-транзисторов и N последовательно включ p-МОП-транзисторов. Последовательно включ транзисторы передают сигнал медленнее, чем включенные параллельно транзисторы. Также, p-МОП-транзисторы передают сигналы медленнее, чем n-МОП-транзисторы, тк дырки не могут перемещаться по кристаллич решетке кремния так же быстро, как электроны. В результате, соединенные параллельно n-МОП-транзисторы работают быстро, а соединенные последовательно p-МОП-транзисторы работают медленно.

3)Потребляемая мощность – это кол-во энергии, потребляемой системой в единицу времени. Цифровая система потребляет энергию как в динамич режиме, так и в статическом. В динамич режиме энергия расходуется на зарядку емкостей элементов системы, когда эти эл переключаются между 0 и 1. И хотя в статическом режиме никаких переключений не происходит, система все равно расходует электрическую энергию.

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

Принципиальная схема – это изображение цифровой схемы, показывающее элементы и соединяющие их проводники. Изображая принципиальные схемы в унифицированном виде, нам становится легче читать их и отлаживать. В большинстве случаев мы будем придерживаться следующих правил: Входы изображаются на левой (или верхней) части схемы; Выходы изображаются на правой (или нижней) части схемы; Всегда, когда это возможно, элементы необходимо изображать слева направо; Проводники лучше изображать прямыми линиями; Проводники всегда должны соединяться в виде буквы «Т»; Точка в месте пересечения проводников обозначает их соединение;

Символ «X» используется не только для обозначения переменных, но и для обозначения недопустимых состояний сигналов при симуляции логических схем. Булева алгебра ограничена значениями 0 и 1. Однако реальные схемы могут также иметь недопустимое и плавающее состояния, представляемые символами 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) и 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. Типично неправильное представление – что неподключенная, или плавающая цепь имеет значение логического соответственно.

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

**Задержка распространения комбинационной схемы** – это сумма задержек распространения всех элементов в критическом пути. **Задержка реакции** – сумма задержек реакции всех элементов в кратчайшем пути.

**Задержка распространения tpd** – это максимальное время от начала изменения входа до момента, когда все выходы достигнут установившихся значений.(сумма задержек распространения всех элементов в критическом пути);**Задержка реакции tcd** – это минимальное время от момента, когда вход изменился, до момента, когда любой из выходов начнет изменять свое значение (сумма задержек реакции всех элементов в кратчайшем пути). **Основные причины задержек в схемах** заключаются во времени, требуемом для перезарядки емкостей цепи, а так же в конечной скорости распространения электромагнитных волн в среде. Величины tpd и tcd могут различаться по многим причинам, включающим в себя: Разные задержки нарастания и спада сигнала; Несколько входов и выходов, одни из которых быстрее чем другие; Замедление работы схемы при повышении температуры и ускорение при охлаждении. **Временные характеристики мультикомплексора**: Критических путь; Задержки распространения и реакции; Импульсные помехи (явление одиночного изменения на входе вызывает несколько выходных изменений);Резюме (Цифровая схема – это модуль с дискретными значениями входов и выходов и спецификацией, описывающей его функциональные и временные характеристики. Функциональное описание комбинационной схемы может быть задано таблицей истинности или логическим выражением. Логическое выражение для любой таблицы истинности может быть получено в виде совершенной дизъюнктивной нормальной формы или совершенной конъюнктивной нормальной формы); **Импульсной помехой или паразитным импульсом** - одиночное изменение на входе вызывает несколько выходных изменений.

**50.Базовые комбинационные блоки. Мультиплексоры. Логика на мультиплексорах. Дешифраторы**

**Мультиплексор** - логическое устройство, предназначенное для **поочерёдной** передачи на один выход одного из нескольких входных сигналов, т.е. их мультиплексирование. **Количество мультиплексируемых входов** от 2 до 16. **Количество выходов** от 1 до 4. **Управление** осуществляется при помощи входного кода. Связь между **n**(кол-во каналов) и **m**(число разрядов) **n = 2^m**. **Функции** записываются как **MS**.**Разряды 1,2,4** и т.д. Входная информация 0,1,2,3,4… **Выходы** бывают **прямыми и инверсными**. Применяют для операций **коммутации, преобразование параллельного кода в логический.** Мультиплексоры могут использоваться как таблицы преобразования (lookup tables) для выполнения логических функций. **Дешифратор -** логическая комбинационная схема,имеющая n входов и 2^n выходов.Если имеется комбинация входных сигналов,то появляется сигнал на том контакте, который соответствует коду,который был на входе. Особенностью двоичного дешифратора является то, что логический сигнал появляется только на выходе, соответствующем номеру двоичной комбинации. Существует несколько разновидностей дешифраторов: прямоугольные, матричные, пирамидальные.

**51.Проектирование последовательности логики. Защелки и триггеры. RS-триггер. D-защелка. D-Триггер. Регистр.**

**Защелка** – элемент, который запоминает свое состояние. **Триггер** – электронная схема обладающая двумя устойчивыми состояниями. Изменение состояния происходит скачкообразно под воздействием управляющих сигналовТриггеры служат основой для построения регистров , счетчиков и других элементов обладающих функцией хранения. Главной частью триггера является запоминающая ячейка. Их разделяют по способу управления (RS, JK. D…), типу синхронизации (синхронные (статическая/динамическая) синхронизация), асинхронные), по внутренней организации (1-/2- ступенчатые). Триггер называется синхронным, если таблица переходов хотя бы по одному управляющему входу осуществляется под воздействием синхронизирующего сигнала. Асинхронные **RS**(по названию входов Reset (установить выход в 0) и Set (установить в 1)) **триггеры** одни из самых простых и часто являются основой для построения более сложных. Триггеры этого типа построены на двух типах логических элементов: ИЛИ-НЕ – триггер с прямым входом, И-НЕ – триггер с прямым входом. **D-триггер** (от слова delay) – триггер с одним входом, работающий так, что сигнал на выходе после переключения равен сигналу на входе D до переключения. **D-защелка** – D триггер, работающий по сигналу на входе C

**Регистр** – устройство для записи хранения считывания и выполнения операций над n-разрядными двоичными данными. Фактически любое цифровое устройство можно представить как совокупность регистров объединенных при помощи комбинационных логических устройств, а сам регистр как упорядоченный набор триггеров. 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.

**52.Триггер с функцией разрешения. Триггер с функцией сброса. Проектирование синхронных логических схем. Синхронные последовательные схемы. Синхронные и асинхронные схемы.**

У **триггеров с функцией разрешения** имеется еще один вход, называемый EN (ENABLE т.е. разрешить). Этот вход определяет, будут ли данные загружаться по фронту. Когда на EN подается логическая 1, то такой D-триггер поведет себя как и обычный D-триггер. Если же поступает логический 0, то триггер игнорирует тактовый сигнал и сохраняет свое состояние. Такие триггеры полезны, если мы хотим загружать значения в триггер только на протяжении определенного времени. В **триггере с функцией сброса** добавляется вход, называемый RESET (сброс). Когда на RESET подан 0, сбрасываемый триггер едет себя как обычный D-триггер. Когда же поступает единица, триггер игнорирует вход D и сбрасывает выход к 0. Такие триггеры полезны, если мы хотим ускорить установления определенного состояния (0) во всех триггерах системы при первом включении. Такие триггеры могут сбрасываться как синхронно, так и асинхронно. В первом случае триггеры сбрасываются только по фронту сигнала CLK. Во втором сброс происходит сразу при поступлении логической единицы на вход RESET, вне зависимости от тактового сигнала.

Схема является **синхронной последовательностной**, если ее элементы удовлетворяют следующим условиям: как минимум один элемент схемы является регистром, остальные – комбинационной схемой, все регистры тактируются единственным тактовым сигналом, в каждом циклическом пути есть как минимум 1 регистр. Последовательностные схемы, не являющиеся **синхронными**, называются **асинхронными**. Асинхронные схемы иногда используются для связи между собой систем с разными тактовыми сигналами или для считывания значений со входов в произвольное время

**53.Конечные автоматы. Пример проектирования конечного автомат**

**Конечный автомат**- это модель вычислений, основанная на гипотетической машине состояний. В один момент времени только одно состояние может быть активным. Следовательно, для выполнения каких-либо действий машина должна менять свое состояние. Конечные автоматы обычно используются для организации и представления потока выполнения чего-либо. Это особенно полезно при реализации ИИ в играх. 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. Например, для написания «мозга» врага: каждое состояние представляет собой какое-то действие (напасть, уклониться и т. д.).

**54. Конечные автоматы. Кодирование состояний. Автоматы Мура и Мили.**

**Конечный автомат**- это модель вычислений, основанная на гипотетической машине состояний. В один момент времени только одно состояние может быть активным. Следовательно, для выполнения каких-либо действий машина должна менять свое состояние. Конечные автоматы обычно используются для организации и представления потока выполнения чего-либо. Это особенно полезно при реализации ИИ в играх. **Кодирование состояний.** Одно из важных решений в кодировании состояний – выбор между двоичным кодированием (00, 01, 10) и прямым кодированием (001, 010, 100)( для каждого состоянияиспользуется один бит состояния), которое также называется кодированием «1 из *N*». При *двоичном кодировании*,как в примере сконтроллером светофора, каждому состоянию ставится в соответствие двоичное число (номер этого состояния). Так как K двоичных чисел можно записать в log2*K* разрядах, системе с *K* состояниями нужно всегоlog2*K* битовсостояния. 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 . **Автомат Мура** в [теории вычислений](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B9) — [конечный автомат](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82), выходное значение сигнала в котором зависит лишь от текущего состояния данного автомата, и не зависит напрямую. **Автомат Мили** — [конечный автомат](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D1%8B%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82), выходная последовательность которого (в отличие от [автомата Мура](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82_%D0%9C%D1%83%D1%80%D0%B0)) зависит от состояния автомата и входных сигналов. у автомата Мура обычно больше состояний, чем у автомата Мили, решающего ту же задачу.

**55.Декомпозиция конечных автоматов. Восстановление конечных автоматов по электрической схеме.**

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. **Восстановление конечных автоматов по электрической схеме** практически является процессом, обратным проектированию КА. Этот процесс необходим, например, при рассмотрении проекта с неполной документацией или для реверсивного проектирования чьей-то системы: 1)Проанализировать схему, возможные состояния входов, выходов и регистра состояний; 2)Составить выражения для следующего состояния и для выходов; 3)Составить таблицу выходов и таблицу переходов; 4)Вычеркнуть из таблицы переходов состояния, в которые система никогда не попадает; 5) Присвоить имя каждому используемому набору бит-состояний; 6) Переписать таблицы выходов и переходов, используя эти обозначения; 7)Нарисовать диаграмму переходов; 8)Описать словами то, что делает автомат;(На последнем шаге развернуто описать цели и функции автомата, чтобы избежать простого переформулирования каждого перехода из диаграммы переходов.)

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

**Временные характеристики системы**

**Расфазировка тактовых сигналов**. 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. В итоге, расфазировка тактовых импульсов приводит к эффективному увеличению как времени предустановки, так и времени удержания. Это, в свою очередь, приводит к росту потерь на упорядочение и уменьшает время, доступное для обработки данных комбинационной схемой. Чтобы предотвратить такие серьезные нарушения ограничений времени удержания, проектировщик должен ограничивать расфазировку тактовых сигналов. Иногда триггеры специально проектируются медленными (время tccq велико), чтобы избежать проблем времени удержания, даже если расфазировка тактовых сигналов существенна. **Метастабильное состояние**- состояние, когда состояние информационного входа триггера изменяется в течение апертурного времени, на его выходе Q может на некоторое время появиться напряжение в диапазоне от 0 до VDD, то есть в запретной зоне. Такое состояние называется метастабильным. Со временем выход триггера перейдет в стабильное состояние 0 или 1. Однако время разрешения, необходимое для достижения стабильного состояния, не ограничивается. **Синхронизатор** - устройство, 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. Надежность системы обычно измеряют средним временем наработки на отказ (mean time between failures, MTBF). Как понятно из названия, MTBF – это среднее время между отказами системы. Эта величина обратна вероятности сбоя системы за любую заданную секунду: MTBF растет экспоненциально с ростом времени ожидания синхронизатора, Tc. Для большинства систем синхронизатор, который ожидает один период тактового сигнала, обеспечивает достаточную величину MTBF. В высокоскоростных системах может понадобиться ожидание на большее количество периодов тактового сигнала.

**57.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.

**Типы триггеров: RS-триггеры** (триггер, который сохраняет своё предыдущее состояние при неактивном состоянии обоих входов и изменяет своё состояние при подаче на один из его входов активного уровня ); **D-триггер** (запоминает состояние входа и выдаёт его на выход. D-триггеры имеют, как минимум, два входа: информационный **D** и синхронизации **С**. Вход синхронизации С может быть статическим (потенциальным) и динамическим.); **Т-триггеры** (часто называют счётным триггером, так как он является простейшим [счётчиком](https://ru.wikipedia.org/wiki/%D0%A1%D1%87%D1%91%D1%82%D1%87%D0%B8%D0%BA_(%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0)) по модулю); **JK-триггер** (работает так же как RS-триггер, с одним лишь исключением: при подаче логической единицы на оба входа J и K состояние выхода триггера изменяется на противоположное, то есть выполняется операция инверсии (чем он отличается от RS-триггеров с доопределённым состоянием, которые строго переходят в логический ноль или единицу, независимо от предыдущего состояния)); **Классификация триггеров:** Триггеры подразделяются на две большие группы — динамические (управляемый генератор, одно из состояний которого (единичное) характеризуется наличием на выходе непрерывной последовательности импульсов определённой частоты, а другое (нулевое) — отсутствием выходных импульсов. Смена состояний производится внешними импульсами) и статические(устройства, каждое состояние которых характеризуется неизменными уровнями выходного напряжения (выходными потенциалами): высоким — близким к напряжению питания и низким — около нуля. Статические триггеры по способу представления выходной информации часто называют потенциальными.).

**Триггерные схемы классифицируют** также по следующим признакам: числу целочисленных устойчивых; числу уровней; по способу реакции на помехи — прозрачные и непрозрачные. Непрозрачные, в свою очередь, делятся на проницаемые и непроницаемые; по составу логических элементов (триггеры на элементах И-НЕ, ИЛИ-НЕ и др.).

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

**Регистры** представляют собой несколько триггеров соединённых между собой различными способами. Два основных вида регистров: параллельный регистр; последовательный регистр или регистр сдвига. **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); **Регистр сдвига** или **последовательный** **регистр** (англ. Shift Register), представляет собой схему, в которой внутренние триггеры соединены последовательно. Схема работы сдвигового регистра заключается в следующем: по импульсу тактового сигнала происходит сдвиг на один разряд цифрового кода, который записан на входном выводе. У обычных сдвиговых регистров, сдвиг происходит от младших разрядов к старшим, но есть также и **реверсивные сдвиговые регистры**, у которых сдвиг идет, наоборот, от старших разрядов к младшим. Большинство регистров сдвига имеют восемь разрядов, различаются режимами работы, режимами записи, чтения и сдвига, а также типом выходного каскада.

**59. Мультиплексоры и демультиплексоры. Отличия в обозначениях цифровых элементов в разных стандартах.**

**Мультиплексор** – коммутатор логических сигналов, обеспечивающий передачу информации, поступающей по нескольким входным линиям связи, на одну выходную линию. Мультиплексор состоит из дешифратора адреса входной линии, схем И и схемы объединения ИЛИ. Двоичный код, воздействующий на адресные входа, откроет одну из схем И, которая соединит с выходом соответствующую входную линию. При этом информация на выходе определяется состоянием выбранного входного канала и не зависит от состояния других каналов. Мультиплексоры можно использовать для синтеза логических функций от нескольких переменных (x1, x2, …, xn). Если число адресных входов мультиплексора m(адр) , то из общего числа n переменных функции m(адр) можно подать на адресные входы. Тогда на информационные входы мультиплексора через дополнительную логическую схему подаются n-m(адр) переменных.

**Демультиплексор** – коммутатор логических сигналов, обеспечивающий передачу информации, поступающей по одному входу, и в соответствии с адресом направляющий в одну из выходных линий.

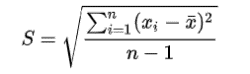
При передачах данных по общему каналу с разделением во времени нужны не только мультиплексоры, но и демультиплексоры, распределяющие данные из одного канала между несколькими приемниками информации.

Демультиплексор имеет один информационный вход n адресующих (управляющих) входов и 2n выходов. Работа демультиплексора описывается логическими выражениями:, где mi – минтермы n адресующих переменных.

Применительно к мультиплексорам и демультиплексорам пользуются также термином «селекторы данных».

**60. Погрешность математических операций в цифровых системах. Способы оценки. Округление.**

**Абсолютной погрешностью** или, погрешностью приближенного числа называется разность между этим числом и его точным значением (из большего числа вычитается меньшее). **Относительной погрешностью** приближенного числа называется отношение абсолютной погрешности приближенного числа к самому этому числу. В большинстве случаев невозможно узнать точное значение приближенного числа, а значит, и точную величину погрешности. Однако почти всегда можно установить, что погрешность (абсолютная или относительная) не превосходит некоторого числа. Число, заведомо превышающее абсолютную погрешность (или в худшем случае равное ей), называется **предельной абсолютной погрешностью**. Число, заведомо превышающее относительную погрешность (или в худшем случае равное ей), называется **предельной относительной погрешностью.** **Округление** применяется для представления значений и результатов вычислений с тем количеством знаков, которое соответствует реальной точности измерений или вычислений, либо той точности, которая требуется в конкретном приложении. Округление в ручных расчётах также может использоваться для упрощения вычислений в тех случаях, когда погрешность, вносимая за счёт ошибки округления, не выходит за границы допустимой погрешности расчёта. **Оценка погрешности**. В зависимости от характеристик измеряемой величины для определения погрешности измерений используют различные методы. Часто для оценки случайной погрешности используют стандартное отклонение, или среднеквадратическое отклонение, для которого обычно используют один из двух способов оценки (оба термина применяются как к одному, так и к другому способу):На основании несмещённой оценки дисперсии:



На основании смещённой оценки дисперсии:



**61. Архитектура процессора. Основные компоненты. Способы классификации. Много уровневая организация. Контроллеры вввода-вывола.**

Aрхитектура процессора

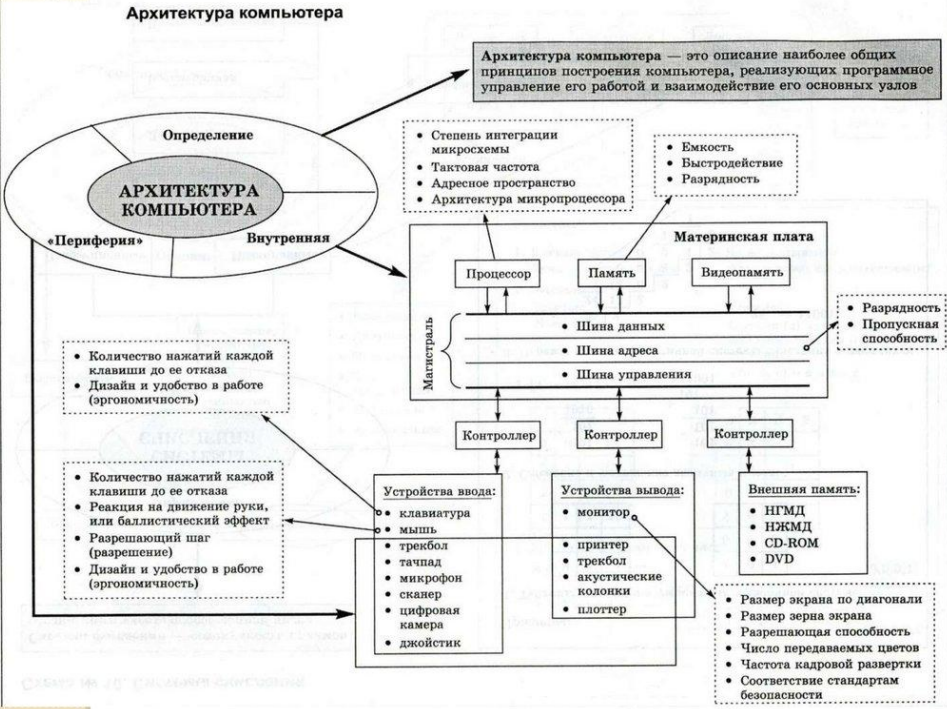
Архитекту́ра проце́ссора — количественная составляющая компонентов микроархитектуры вычислительной машины (процессора компьютера) (например, регистр флагов или регистры процессора), рассматриваемая IT-специалистами в аспекте прикладной деятельности.

С точки зрения:

• программиста — совместимость с определённым набором команд (например, процессоры, совместимые с командами Intel x86), их структуры (например, систем адресации или организации регистровой памяти) и способа исполнения (например, счётчик команд).

• аппаратной составляющей вычислительной системы — это некий набор свойств и качеств, присущий целому семейству процессоров (иначе говоря — «внутренняя конструкция», «организация» этих процессоров).

• 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).



Контроллеры ввода-вывода

• Super I/O (англ. Super Input/output) — название класса сопроцессоров, которые начали использоваться после 1980-х годов на материнских платах IBM PC-совместимых компьютеров путём сочетания функций многих контроллеров, сперва одной платой, устанавливаемой в слот расширения, а затем и одной микросхемой, тем самым достигая уменьшения числа микросхем контроллеров, и таким образом привели к снижению сложности и стоимости компьютера в целом. Super I/O объединяет интерфейсы различных низкоскоростных устройств.

• Фактически, на рынке материнских плат массового сегмента предоставлена продукция всего четырёх фирм: ITE Tech, Nuvoton, Microchip Technology и Fintek. Производители брендовой продукции (IBM, HP, Dell, FSC) как правило используют проприетарные разработки схемотехники материнских плат и не публикуют спецификаций применяемых контроллеров.

• 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).

• Super I/O также может включать в себя и другие интерфейсы, такие как игровой (MIDI или джойстик) или инфракрасный порты.

• контроллер Ethernet

• Изначально Super I/O связывались через шину ISA. Одновременно с развитием IBM PC совместимых компьютеров происходило смещение Super I/O, сперва на шины VLB, затем стала использоваться шина PCI. Современные Super I/O используют шину LPC (интерфейс которой предоставляет южный мост материнской платы) и часто реализованы в составе чипсета.

• Схемотехника материнской платы предполагает наличие цепей измерения, которые производятся с помощью аналогово-цифровых преобразователей, преобразующих измеряемый параметр в цифровые значения, после чего они могут быть переданы в другое вычислительное устройство на плате для дальнейшей обработки. Измерения, производимые на материнской плате, в основном касаются трёх групп параметров: обороты вентиляторов, температура и напряжения.

• Скорость вращения вентиляторов, применяемых для охлаждения блоков и отдельных частей материнской платы, обычно контролируется при помощи тахометров, встроенных в вентилятор — обычно для этого используется датчик Холла. Такой вентилятор отличается дополнительными (кроме двух питания) проводами, одним (устанавливаемый в корпус, блок питания, на охлаждение радиаторов микросхем чипсепа материнской платы/видеокарты) или двумя (процессорный).

• Для мониторинга температуры используется три типа датчиков: терморезисторы, транзисторы (например, 2N3904) и датчики интегрированные в процессор.

• Логически аппаратный мониторинг выглядит как набор регистров, значение которых изменяется при изменении состояния на входах.

• USB-контроллер в составе платформы персонального компьютера обеспечивает коммуникацию с периферийными устройствами, подключенными к USB. USB-контроллер является устройством, способным взаимодействовать с оперативной памятью в обход центрального процессора в режиме прямого доступа к памяти.

• По способу интеграции контроллер для USB-шины может быть задействован в составе системной логики или в виде дискретного чипа как на самой системной плате, так и на плате расширения. По способу подключения USB-контроллер может быть выполнен для PCI-шины, либо для шины PCI Express.





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

RICS

• 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.

• По сравнению с CISC эта архитектура имеет константную длину команды, а также меньшее количество схожих инструкций, позволяя уменьшить итоговую цену процессора и энергопотребление, что критично для мобильного сегмента. У RISC также большее количество регистров.

• Примеры RISC-архитектур: PowerPC, серия архитектур ARM (ARM7, ARM9, ARM11, Cortex).

• В общем случае RISC быстрее CISC. Даже если системе RISC приходится выполнять 4 или 5 команд вместо одной, которую выполняет CISC, RISC все равно выигрывает в скорости, так как RISC-команды выполняются в 10 раз быстрее.

• Отсюда возникает закономерный вопрос: почему многие всё ещё используют CISC, когда есть RISC? Всё дело в совместимости. x86\_64 всё ещё лидер в desktop-сегменте только по историческим причинам. Так как старые программы работают только на x86, то и новые desktop-системы должны быть x86(\_64), чтобы все старые программы и игры могли работать на новой машине.

• Для Open Source это по большей части не является проблемой, так как пользователь может найти в интернете версию программы под другую архитектуру. Сделать же версию проприетарной программы под другую архитектуру может только владелец исходного кода программы.

CISC

• 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.

• Самый яркий пример CISC архитектуры — это x86 (он же IA-32) и x86\_64 (он же AMD64).

• В CISC процессорах одна команда может быть заменена ей аналогичной, либо группой команд, выполняющих ту же функцию. Отсюда вытекают плюсы и минусы архитектуры: высокая производительность благодаря тому, что несколько команд могут быть заменены одной аналогичной, но большая цена по сравнению с RISC процессорами из-за более сложной архитектуры, в которой многие команды сложнее раскодировать

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.

• Для увеличения производительности во всех вышеперечисленных архитектурах может использоваться “спекулятивное исполнение команд”. Это выполнение команды до того, как станет известно, понадобится эта команда или нет.

ARM

• Архитектура (от англ. Advanced RISC Machine — усовершенствованная RISC машина; иногда — Acorn RISC Machine) — система команд и семейство описаний и готовых топологий 32-битных и 64- битных микропроцессорных/микроконтроллерных ядер, разрабатываемых компанией ARM Limited

• Среди лицензиатов готовых топологий ядер ARM — компании AMD, Apple, Analog Devices, Atmel, Xilinx, Cirrus Logic[en], Intel (до 27 июня 2006 года), Marvell, NXP, STMicroelectronics, Samsung, LG, MediaTek, Qualcomm, Sony, T exas Instruments, Nvidia, Freescale, Миландр, ЭЛВИС[2], HiSilicon, Байкал электроникс.

• Значимые семейства процессоров: ARM7, ARM9, ARM11 и Cortex.

• В 2006 году около 98 % из более чем миллиарда мобильных телефонов, продававшихся ежегодно, были оснащены, по крайней мере, одним процессором ARM[5]. По состоянию на 2009, на процессоры ARM приходилось до 90 % всех встроенных 32-разрядных процессоров[6]. Процессоры ARM широко используются в потребительской электронике — в том числе смартфонах, мобильных телефонах и плеерах, портативных игровых консолях, калькуляторах, умных часах и компьютерных периферийных устройствах, таких, как жесткие диски или маршрутизаторы.

• Многие лицензиаты проектируют собственные топологии ядер на базе системы команд ARM

• Эти процессоры имеют низкое энергопотребление, поэтому находят широкое применение во встраиваемых системах и преобладают на рынке мобильных устройств, для которых данный фактор немаловажен.

• В основном процессоры семейства завоевали сегмент массовых мобильных продуктов (сотовые телефоны, карманные компьютеры) и встраиваемых систем средней и высокой производительности (от сетевых маршрутизаторов и точек доступа до телевизоров). Отдельные компании заявляют о разработках эффективных серверов на базе кластеров ARM-процессоров, но пока это только экспериментальные проекты с 32-битной архитектурой

• Уже давно существует справочное руководство по архитектуре ARM, которое разграничивает все типы интерфейсов, которые поддерживает ARM, так как детали реализации каждого типа процессора могут различаться. Архитектура развивалась с течением времени и, начиная с ARMv7, были определены 3 профиля:

• A (application) — для устройств, требующих высокой производительности (смартфоны, планшеты);

• R (real time) — для приложений, работающих в реальном времени;

• M (microcontroller) — для микроконтроллеров и недорогих встраиваемых устройств.

• Профили могут поддерживать меньшее количество команд (команды определенного типа).

• Набор команд

• Чтобы сохранить устройство чистым, простым и быстрым, оригинальное изготовление ARM было исполнено без микрокода, как и более простой 8-разрядный процессор 6502, используемый в предыдущих микрокомпьютерах от Acorn Computers.

• Набор команд ARM. Режим, в котором исполняется 32-битный набор команд. ADC, ADD, AND, B/BL, BIC, CMN, CMP, EOR, LDM, LDR/LDRB, MLA, MOV, MUL, MVN, ORR, RSB, RSC, SBC, STM, STR/STRB, SUB, SWI, SWP, TEQ, TST

• Набор команд Thumb. Для улучшения плотности кода процессоры, начиная с ARM7TDMI, снабжены режимом «thumb». В этом режиме процессор выполняет альтернативный набор 16- битных команд. Большинство из этих 16-разрядных команд переводится в нормальные команды ARM. Уменьшение длины команды достигается за счёт сокрытия некоторых операндов и ограничения возможностей адресации по сравнению с режимом полного набора команд ARM.

• В режиме Thumb меньшие коды операций обладают меньшей функциональностью. Например, только ветвления могут быть условными, и многие коды операций имеют ограничение в виде доступа только к половине главных регистров процессора.

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. Компиляторы для процессоров этой архитектуры сильно привязаны к конкретным процессорам. Например, в следующем поколении максимальная длина «очень длинной команды» может из условных 256 бит стать 512 бит, и тут приходится выбирать между увеличением производительности путём компиляции под новый процессор и обратной совместимостью со старым процессором. Опять же, Open Sourсe позволяет простой перекомпиляцией получить программу под конкретный процессор. Примеры архитектуры: Intel Itanium, Эльбрус-3.

• 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.

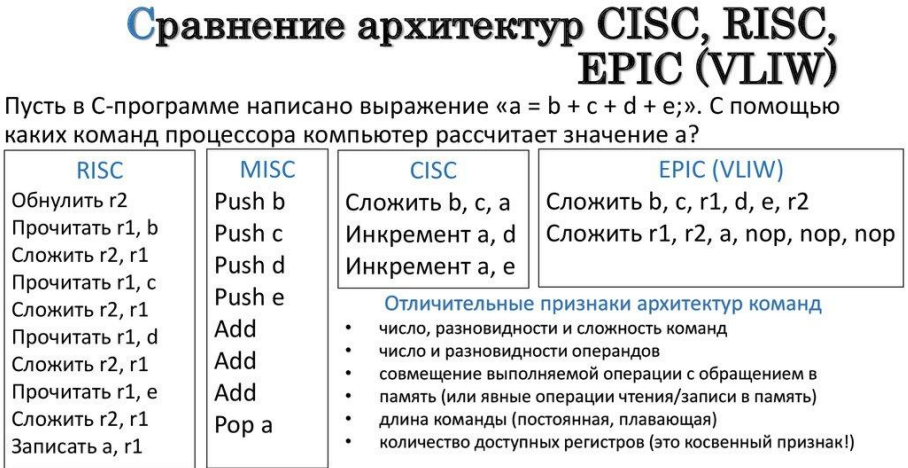
• Уязвимость потенциально позволяет локальным приложениям (локальному атакующему, при запуске специальной программы) получить доступ к содержимому виртуальной памяти текущего приложения или других программ Ошибка Spectre позволяет злонамеренным пользовательским приложениям, работающим на данном компьютере, получить доступ на чтение к произвольным местам компьютерной памяти, используемой процессом-жертвой, например другими приложениями (то есть нарушить изоляцию памяти между программами). Атаке Spectre подвержено большинство компьютерных систем, использующих высокопроизводительные микропроцессоры, в том числе персональные компьютеры, серверы, ноутбуки и ряд мобильных устройств[7]. В частности, атака Spectre была продемонстрирована на процессорах производства корпораций Intel, AMD и на чипах, использующих процессорные ядра ARM. Имеется вариант атаки Spectre, использующий JavaScript программы для получения доступа к памяти браузеров (чтение данных других сайтов или данных, сохраненных в браузере)

• 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[1][2]), из-за которой при спекулятивном выполнении инструкций чтения из памяти процессор игнорирует права доступа к страницам.

• Уязвимость позволяет локальному атакующему (при запуске специальной программы) получить несанкционированный доступ на чтение к привилегированной памяти (памяти, используемой ядром операционной системы)

**63. Сравнительная характеристика архитектур. В чем преимущества. Преимущества RISK. Какова проблема лицензирования архитектур.**





Сравнительная оценка cisc- resc- vliw- архитектур



Преимущества RISC

• С одной стороны писать на Assembler'е под RISC процессоры не очень-то удобно. Если в лоб сравнивать код, написанный под CISC и RISC процессоры, очевидно преимущество первого. Так выглядит код одной и той же операции для x86 и ARM. x86

• MOV AX, 15; AH = 00, AL = 0Fh

• AAA; AH = 01, AL = 05

• RET

• ARM

• MOV R3, #10

• AND R2, R0, #0xF

• CMP R2, R3

• IT LT

• BLT elsebranch

• ADD R2. #6

• ADD R1. #1

• elsebranch:

• END

• Каждый цветной квадрат — это отдельные команды. Их много и они разные. Как вы поняли, здесь мы уже говорим про микроархитектуру, которая вытекает из набора команд. А вот ARM-процессор скорее выглядит так.

• Ему не нужны блоки, созданные для функций, написанных 50 лет назад. По сути, тут блоки только для самых востребованных команд. Зато таких блоков много. А это значит, что можно одновременно выполнять больше базовых команд. А раритетные не занимают место.

• Но так было раньше. На ассемблере уже давно никто не пишет. Сейчас за программистов всё это делают компиляторы, поэтому никаких сложностей с написанием кода под RISC-процессоры нет. Зато есть преимущества. Представьте, что вы проектируете процессор. Расположение блоков на х86 выглядело бы так.

Лицензирование

• Но это все отличия технические. Есть отличия и организационные. Вы не задумывались почему для смартфонов так много производителей процессоров, а в мире ПК на x86 только AMD и Intel? Все просто — ARM это компания которая занимается лицензированием, а не производством. Даже Apple приложила руку к развитию ARM. Вместе с Acorn Computers и VLSI Technology. Apple присоединился к альянсу из-за их грядущего устройства — Newton. Устройства, главной функцией которого было распознавание текста. Даже вы можете начать производить свои процессоры, купив лицензию. А вот производить процессоры на x86 не может никто кроме синей и красной компании. А это значит что? Правильно, меньше конкуренции, медленнее развитие. Как же так произошло?

• Ну окей. Допустим ARM прекрасно справляется со смартфонами и планшетами, но как насчет компьютеров и серверов, где вся поляна исторически поделена? И зачем Apple вообще ломанулась туда со своим Apple Silicon.

Что сейчас?

• Допустим мы решили, что архитектура ARM более эффективная и универсальная. Что теперь? x86 похоронен? На самом деле, в Intel и AMD не дураки сидят. И сейчас под капотом современные CISC-процессоры очень похожи на RISC. Постепенно разработчики CISC-процессоров все-таки пришли к этому и начали делать гибридные процессоры, но старый хвост так просто нельзя сбросить.

• Но уже достаточно давно процессоры Intel и AMD разбивают входные инструкции на более мелкие микро инструкции (micro-ops), которые в дальнейшем — сейчас вы удивитесь — исполняются RISC ядром.

• Те самые 4-8 ядер в вашем ПК — это тоже RISC-ядра!

• Надеюсь, тут вы окончательно запутались. Но суть в том, что разница между RISC и CISC-дизайнами уже сейчас минимальна. А что остается важным — так это микроархитектура. То есть то, насколько эффективно все организовано на самом камне. Ну вы уже наверное знаете, что Современные iPad практически не уступают 15- дюймовым MacBook Pro с процессорами Core i7 и Core i9.

• Еще один бонус сокращенного набора RISC: меньше места на чипе занимает блок по декодированию команд. Да, для этого тоже нужно место. Архитектура RISC проще и удобнее, загибайте пальцы:

• проще работа с памятью,

• более богатая регистровая архитектура,

• легче делать 32/64/128 разряды,

• легче оптимизировать,

• меньше энергопотребление,

• проще масштабировать и делать отладку.

• Для примера вот два процессора одного поколения. ARM1 и Intel 386. При схожей производительности ARM вдвое меньше по площади. А транзисторов на нем в 10 раз меньше: 25 тысяч против 275 тысяч. Энергопотребление тоже отличается на порядок: 0.1 Ватт против 2 Ватт у Intel. Шок. Поэтому наши смартфоны, которые работают на ARM процессорах с архитектурой RISC, долго живут, не требуют активного охлаждения и такие быстрые.

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

Виртуальные архитектуры

• Но раз нельзя запустить программу одной архитектуры на другой, то откуда берутся 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. Этот минус нивелируется с помощью JIT- и AOTкомпиляции. Однако большим плюсом будет кроссплатформенность.

• Дальнейшим развитием этих архитектур стали гибридные архитектуры. Например современные x86\_64 процессоры хотя и CISC-совместимы, но являются процессорами с RISC-ядром. В таких гибридных CISC-процессорах CISC-инструкции преобразовываются в набор внутренних RISC-команд. Какое дальнейшее развитие получат архитектуры процессора, покажет только время.

Команды (инструкции)

• 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.

• Эти же инструкции делятся на другие подтипы в зависимости от того, где находятся данные. Например, некоторые инструкции позволяют работать с данными, находящимися в регистрах в этот момент, в то время как в других случаях мы должны отметить адрес памяти, в котором находятся данные (прямой режим), или адрес адреса памяти (косвенный режим ).

• 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.

• Прим. перев. На самом деле все арифметические операции в АЛУ могут быть созданы на основе всего двух: сложение и сдвиг. Однако чем больше базовых операций поддерживает АЛУ, тем оно быстрее.

• Инструкции предоставляются компьютеру на языке ассемблера или генерируются компилятором высокоуровневых языков.

• В процессоре инструкции реализуются на аппаратном уровне. За один такт одноядерный процессор может выполнить одну элементарную (базовую) инструкцию.

• Группу инструкций принято называть набором команд (англ. instruction set).

Тактирование процессора

• 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.

•Тактовые сигналы создают ещё одну сложность при проектировании процессора: поскольку их частоты постоянно растут, то на работу начинают влиять законы физики. Даже несмотря на чрезвычайно высокую скорость света, она недостаточно велика для высокопроизводительных процессоров. Если подключить тактовый сигнал к одному концу чипа, то ко времени, когда сигнал достигнет другого конца, он будет рассинхронизован на значительную величину. Чтобы синхронизировать все части чипа, тактовый сигнал распределяется при помощи так называемого H-Tree. Это структура, гарантирующая, что все конечные точки находятся на совершенно одинаковом расстоянии от центра.

• Может показаться, что проектирование каждого отдельного транзистора, тактового сигнала и контакта питания в чипе — чрезвычайно монотонная и сложная задача, и это в самом деле так. Даже несмотря на то, что в таких компаниях, как Intel, Qualcomm и AMD, работают тысячи инженеров, они не смогли бы вручную спроектировать каждый аспект чипа. Для проектирования чипов такого масштаба они используют множество сложных инструментов, автоматически генерирующих конструкции и электрические схемы. Такие инструменты обычно получают высокоуровневое описание того, что должен делать компонент, и определяют наилучшую аппаратную конфигурацию, удовлетворяющую этим требованиям. Недавно возникло направление развития под названием High Level Synthesis, которое позволяет разработчикам указывать необходимую функциональность в коде, после чего компьютеры определяют, как оптимальнее достичь её в оборудовании.

• Точно так же, как вы можете описывать компьютерные программы через код, проектировщики могут описывать кодом аппаратные устройства. Такие языки, как Verilog и VHDL позволяют проектировщикам оборудования выражать функциональность любой создаваемой ими электрической схемы. После выполнения симуляций и верификации таких проектов их можно синтезировать в конкретные транзисторы, из которых будет состоять электрическая схема. Хоть этап верификации может и не кажется таким увлекательным, как проектирование нового кэша или ядра, он значительно важнее их. На каждого нанимаемого компанией инженера-проектировщика может приходиться пять или более инженеров по верификации.

• Такие компании, как Intel, AMD и Nvidia, не публикуют схем работы своих процессоров, поэтому невозможно показать подобных полных электрических схем для современных процессоров. Однако этот простой сумматор позволит вам получить представление о том, что даже самые сложные части процессора можно разбить на логические и запоминающие элементы, а затем и на транзисторы. Теперь, когда мы знаем, как производятся некоторые компоненты процессора, нам нужно разобраться, как соединить всё вместе и синхронизировать. Все ключевые компоненты процессора подключены к синхронизирующему (тактовому) сигналу (clock signal). Он попеременно имеет высокое и низкое напряжение, меняя его с заданным интервалом, называемым частотой (frequency). Логика внутри процессора обычно переключает значения и выполняет вычисления, когда синхронизирующий сигнал меняет напряжение с низкого на высокое. Синхронизируя все части, мы можем гарантировать, что данные всегда поступают в правильное время, чтобы в процессоре не возникали «глюки». Вы могли слышать, что для повышения производительности процессора можно увеличить частоту тактовых сигналов. Это повышение производительности происходит благодаря тому, что переключение транзисторов и логики внутри процессора начинает происходить чаще, чем предусмотрено. Поскольку в секунду происходит больше циклов, то можно выполнить больше работы и процессор будет иметь повышенную производительность. Однако это справедливо до определённого предела. Современные процессоры обычно работают с частотой от 3,0 ГГц до 4,5 ГГц, и эта величина почти не изменилась за последние десять лет. Точно так же, как металлическая цепь не прочнее её самого слабого звена, процессор может работать не быстрее его самой медленной части. К концу каждого тактового цикла каждый элемент процессора должен завершить свою работу. Если какие-то части ещё её не завершили, то тактовый сигнал слишком быстрый и процессор не будет работать. Проектировщики называют эту самую медленную часть критическим путём (Critical Path) и именно он определяет максимальную частоту, с которой может работать процессор. Выше определённой частоты транзисторы просто не успевают достаточно быстро переключаться и начинают глючить или выдавать неверные выходные значения.

• Повысив напряжение питания процессора, мы можем ускорить переключение транзисторов, но это тоже срабатывает до определённого предела. Если подать слишком большое напряжение, то мы рискуем сжечь процессор. Когда мы повышаем частоту или напряжение процессора, он всегда начинает излучать больше тепла и потреблять бОльшую мощность. Так происходит потому, что мощность процессора прямо пропорциональна частоте и пропорциональна квадрату напряжения. Чтобы определить энергопотребление процессора, мы рассматриваем каждый транзистор как маленький конденсатор, который нужно заряжать или разряжать при изменении его значения. Подача питания — это настолько важная часть процессора, что в некоторых случаях до половины физических контактов на чипе может использоваться только для питания или заземления. Некоторые чипы при полной нагрузке могут потреблять больше 150 амперов, и со всем этим током нужно управляться чрезвычайно аккуратно. Для сравнения: центральный процессор генерирует больше тепла на единицу площади, чем ядерный реактор. Тактовый сигнал в современных процессорах отнимает примерно 30-40% от его общей мощности, потому что он очень сложен и должен управлять множеством различных устройств. Для сохранения энергии большинство процессоров с низким потреблением отключает части чипа, когда они не используются. Это можно реализовать отключением тактового сигнала (этот способ называется Clock Gating) или отключением питания (Power Gating).

• Верификация нового проекта часто занимает больше времени и денег, чем создание самого чипа. Компании тратят так много времени и средств на верификацию, потому что после отправки чипа в производство его невозможно исправить. В случае ошибки в ПО можно выпустить патч, но оборудование работает иначе. Например, компания Intel обнаружила баг в модуле деления с плавающей запятой некоторых чипов Pentium, и в результате это вылилось в потери, эквивалентные современным 2 миллиардам долларов. Сложно осмыслить то, что в одном чипе может быть несколько миллиардов транзисторов и понять, что все они делают. Если разбить чип на его отдельные внутренние компоненты, становится немного легче. Из транзисторов составляются логические элементы, логические элементы комбинируются в функциональные модули, выполняющие определённую задачу, а эти функциональные модули соединяются вместе, образуя архитектуру компьютера, которую мы обсуждали в первой части серии. БОльшая часть работ по проектированию автоматизирована, но изложенное выше позволяет нам осознать, насколько сложен только что купленный нами новый ЦП. Мы обсудили транзисторы, логические элементы, подачу питания и синхронизирующих сигналов, синтез конструкции и верификацию. В третьей части мы узнаем, что требуется для физического производства чипа. Все компании любят хвастаться тем, насколько современен их процесс изготовления (Intel — 10-нанометровый, Apple и AMD — 7-нанометровый, и т.д.), но что же на самом деле означают эти числа? Об этом мы расскажем в следующей части.

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.

• Наборы регистров и инструкций ЦП называются ISA (Архитектура набора инструкций), и все в рамках одного ISA используют одинаковую кодировку инструкций и, следовательно, один и тот же двоичный код для них.

• Ещё в процессоре есть дополнительные регистры, которые хранят в себе последние 4 бита выполненных инструкций.

• Ниже рассмотрен пример набора команд, который суммирует два числа:

• LOAD\_A 8. Это команда сохраняет в ОЗУ данные, скажем, . Первые 4 бита — код операции. Именно он определяет инструкцию. Эти данные помещаются в регистры инструкций УУ. Команда декодируется в инструкцию load\_A — поместить данные 1000 (последние 4 бита команды) в регистр A.

• LOAD\_B 2. Ситуация, аналогичная прошлой. Здесь помещается число 2 (0010) в регистр B.

• ADD B A. Команда суммирует два числа (точнее прибавляет значение регистра B в регистр A). УУ сообщает АЛУ, что нужно выполнить операцию суммирования и поместить результат обратно в регистр A.

• STORE\_A 23. Сохраняем значение регистра A в ячейку памяти с адресом 23.

• Вот такие операции нужны, чтобы сложить два числа.

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.

• Связь набора инструкций с языком ассемблера • Все семейства процессоров имеют общий язык ассемблера, инструкции которого имеют соотношение 1: 1 с набором регистров и инструкций этого семейства процессоров. В приведенной выше таблице вы можете увидеть взаимосвязь между различными инструкциями языка ассемблера x86 и их кодом инструкций, который в таблице выражен в шестнадцатеричном формате.

• Имейте в виду, что в ISA постоянно добавляются новые инструкции, что приводит к появлению очень новых программ, которые явно используют эти новые инструкции, работают только на процессорах, которые их поддерживают. В общем, наборы инструкций стабильны во времени с небольшими изменениями, но время от времени вводятся инструкции для конкретных рынков, которые либо становятся частью стандарта, либо позже отбрасываются.

• Также есть случай, когда новые инструкции более эффективны, чем существующие, но в которых эти инструкции не исключаются из набора, потому что на рынке существует большое количество программного обеспечения, которое зависит от них

**65. Регистр процессора: предназначение, виды. Шины:предназначение, виды.. Кэш: предназначение, виды.**

Регистры 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.

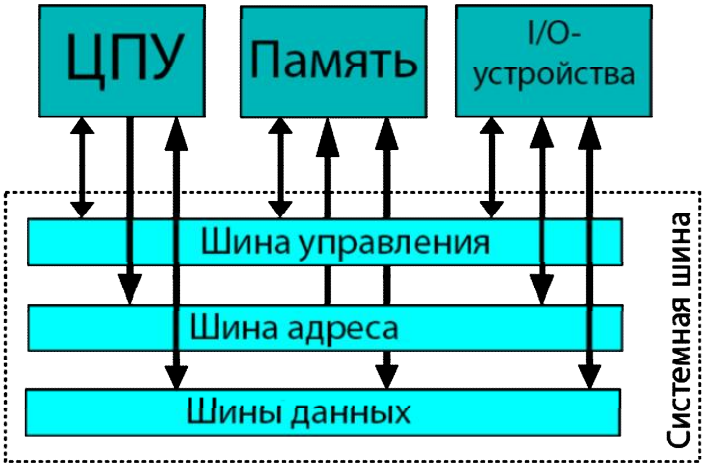
• Некоторые из регистров ЦП, такие как регистр счетчика программ, который указывает, на какой следующий адрес памяти указывает процессор, находятся во всех ЦП и других типах процессоров с возможностью выполнения программ, в то время как другие записи уникальны для каждого набор записей и инструкций, делающий корреляцию 1: 1 между различными ISA практически невозможной.

• Даже если бы у нас был преобразователь кода инструкции 1: 1, у нас все равно были бы проблемы, потому что, хотя два процессора могут иметь одну и ту же инструкцию сложения, мы можем обнаружить, что способ использования регистров и регистров, которые они используют, различны и что есть даже регистры, которые есть в одной семье, а в других нет. Примером этих трудностей столкнулись оба Microsoft и Qualcomm при адаптации Windows 10 в ARM, чтобы все приложения x86 без проблем работали на процессоре ARM.

• Однако есть решения, такие как использование программного обеспечения для перевода инструкций. Указанное программное обеспечение переводит двоичный код в промежуточный код, а затем передает его в двоичный код целевого процессора, в котором мы хотим запустить приложение. Очевидно, что этот процесс намного медленнее, и рекомендуется запускать только очень старое программное обеспечение из семейств процессоров, которые не существуют на рынке

Шина

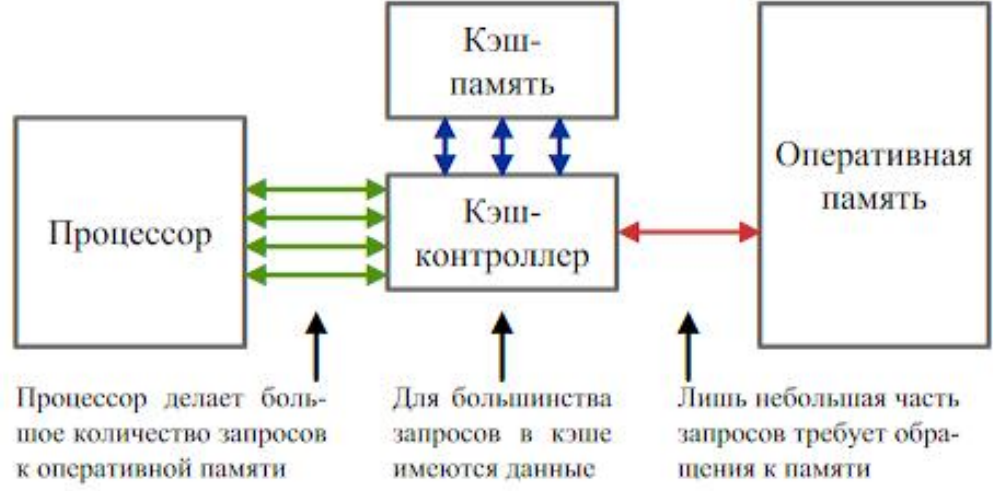
• Все данные между процессором, регистрами, памятью и I/O-устройствами (устройствами ввода-вывода) передаются по шинам. Чтобы загрузить в память только что обработанные данные, процессор помещает адрес в шину адреса и данные в шину данных. Потом нужно дать разрешение на запись на шине управления



Кэш

• Кэш процессора — кэш, используемый процессором компьютера для уменьшения среднего времени доступа к компьютерной памяти. Является одним из верхних уровней иерархии памяти. Кэш использует небольшую, очень быструю память (обычно типа SRAM), которая хранит копии часто используемых данных из основной памяти. У процессора есть механизм сохранения инструкций в кэш. Как мы выяснили ранее, за секунду процессор может выполнить миллиарды инструкций. Поэтому если бы каждая инструкция хранилась в ОЗУ, то её изъятие оттуда занимало бы больше времени, чем её обработка. Поэтому для ускорения работы процессор хранит часть инструкций и данных в кэше.

• Если данные в кэше и памяти не совпадают, то они помечаются грязными битами (англ. dirty bit)



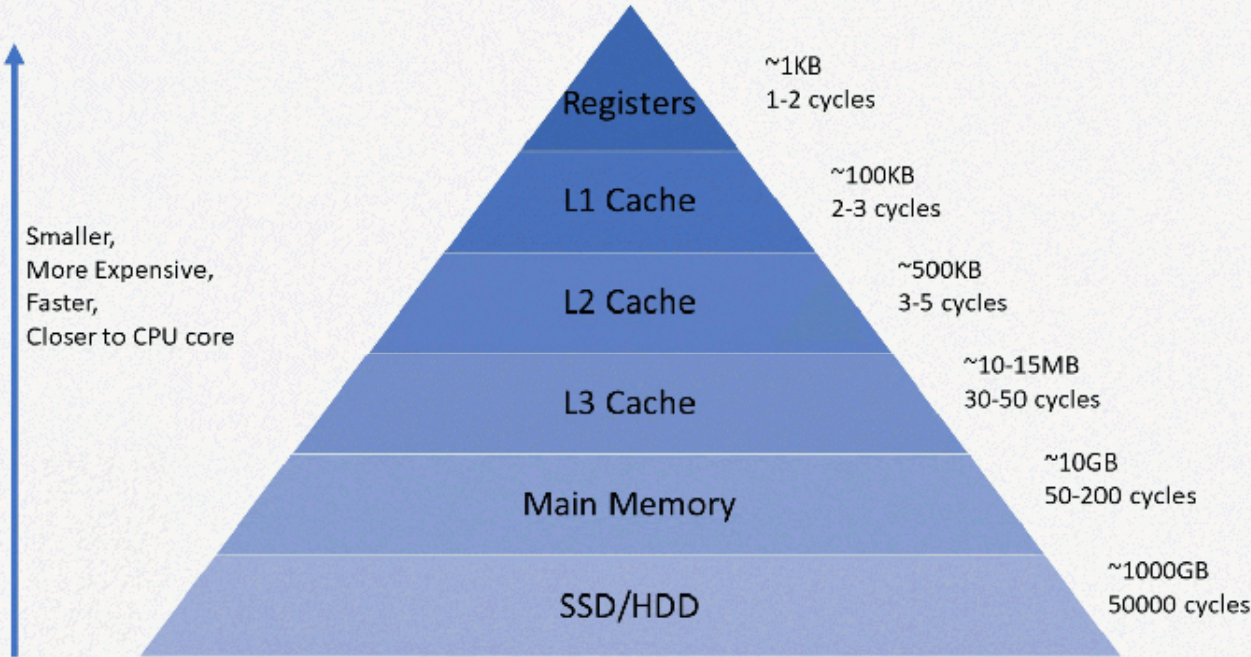
**66. Что такое суперскалярная архитектура. Ее особенности. Предсказатели переходов. Иерархия памяти. Ветвление**

суперскалярная архитектура

• Кроме выполнения с изменением очерёдности современные процессоры применяют технологию под названием **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. Одна из популярных реализаций такой архитектуры называется Simultaneous Multithreading (SMT) и также известна, как HyperThreading. Процессоры Intel и AMD сейчас поддерживают двухсторонний SMT, а IBM разработала чипы, поддерживающие до восьми SMT.

Иерархия памяти

• Когда ЦП запрашивает данные из памяти, то сначала проверяет, хранятся ли эти данные уже в кэше L1. Если да, то можно всего за пару циклов получить к ним доступ. Если их там нет, то процессор проверяет L2, а затем и кэш L3. Кэши реализованы таким образом, что в общем случае они прозрачны для ядра. Ядро просто запрашивает данные по указанному адресу памяти, и тот уровень в иерархии, на котором они есть, отвечает ему. При переходе к последующим уровням в иерархии памяти размер и задержки обычно растут на порядки величин. В конце концов, если ЦП не находит данные ни в одном из кэшей, то обращается в основную память (ОЗУ).



• В обычном процессоре каждое ядро имеет два кэша L1: один для данных и другой для инструкций. Кэши L1 обычно имеют в целом объём порядка 100 килобайт и размер очень варьируется в зависимости от чипа и поколения процессора. Кроме того, обычно для каждого ядра есть свой кэш L2, хотя в некоторых архитектурах он может быть общим для двух ядер. Кэши L2 обычно имеют размер несколько сотен килобайт. Наконец, есть единственный кэш L3, общий для всех ядер, имеющий размер порядка десятков мегабайт. Когда процессор выполняет код, самые часто используемые инструкции и значения данных кэшируются. Это значительно ускоряет выполнение, потому что процессору не нужно постоянно обращаться за нужными данными в основную память. Во второй и третьей частях серии мы подробнее поговорим о том, как реализованы эти системы памяти.

ветвления

Кроме кэшей одним из самых важных строительных блоков современного процессора является точный предсказатель 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.

• На первый взгляд кажется, что эти инструкции ветвления не должны вызывать проблем, но их правильное выполнение может оказаться очень сложным для процессора. В любой момент времени процессор может находиться в процессе одновременного выполнения десяти или двадцати инструкций, поэтому очень важно знать, какие инструкции выполнять. Может потребоваться 5 циклов, чтобы определить, что текущая инструкция — это переход и ещё 10 циклов, чтобы определить истинность условия. В это время процессор уже может начать выполнение десятков дополнительных инструкций, даже не зная, действительно ли это подходящие для выполнения инструкции. Чтобы обойти эту проблему, все современные высокопроизводительные процессоры используют методику под названием «упреждение» (speculation). Это означает, что процессор отслеживает инструкции ветвления и гадает, будет ли выполнен условный переход, или нет. Если предсказание верно, то процессор уже начал выполнять последующие инструкции, и это обеспечивает рост производительности. Если предсказание неверно, то процессор останавливает выполнение, удаляет все неверные инструкции, которые он начал выполнять, и начинает заново с правильной точки.

**Предсказатели перехода**

Такие предсказатели перехода — одни из самых простейших разновидностей машинного обучения, потому что предсказатель изучает поведение ветвей в процессе выполнения. Если он предсказывает неверно слишком часто, то начинает обучаться правильному поведению. Десятилетия исследований методик предсказания переходов привели к тому, что в современных процессорах точность предсказаний превышает 90%

• Хотя упреждение обеспечивает огромный рост производительности, потому что процессор может выполнять инструкции, которые уже готовы, вместо того, чтобы ожидать в очереди завершения выполняемых, оно в то же время создаёт уязвимости в защите. Знаменитая атака Spectre эксплуатирует баги в предсказании и упреждении переходов. Атакующий использует специально подобранный код, чтобы заставить процессор упреждающе выполнить код, благодаря чему происходит утечка значений из памяти. Для предотвращения утечки данных необходимо было переделать конструкцию отдельных аспектов упреждения, что привело к небольшому падению производительности. За последние десятилетия используемая в современных процессорах архитектура прошла долгий путь. Инновации и разработка продуманной структуры привели к повышению производительности и более оптимальному использованию аппаратных средств. Однако разработчики центральных процессоров тщательно хранят секреты их технологий, поэтому мы не можем точно узнать, что происходит у них внутри. Тем не менее, фундаментальные принципы работы процессоров стандартированы для всех архитектур и моделей. Intel может добавлять свои секретные ингредиенты, чтобы повысить долю попаданий кэша, а AMD может добавить улучшенный предсказатель переходов, но процессоры обеих компаний выполняют одинаковую задачу

**67. Что такое гетерагенные вычисления. FPGA-акселератор? Сфера применения. Перспективные направления развития вычислительных систем.**

**Гетерогенные вычисления**

Нарастает тенденция влияния гетерогенных вычислений. 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**

• Но GPU — не единственная область, в которой акселераторы становятся всё популярнее. В большинстве смартфонов есть десятки аппаратных акселераторов, предназначенных для ускорения очень специфических задач. Такой стиль вычислений называется морем акселераторов (Sea of Accelerators), его примерами могут быть криптографические процессоры, процессоры изображений, ускорители машинного обучения, кодеры/декодеры видео, биометрические процессоры и многое другое. Нагрузки становятся всё более и более специализированными, поэтому проектировщики включают в свои чипы всё больше акселераторов. Поставщики облачных услуг, например AWS, начали предоставлять разработчикам FPGAкарты для ускорения их вычислений в облаках. 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.

• К упомянутым отдельным FPGA-акселераторам добавлю такую тему как встраиваемые FPGA (embedded FPGA, eFPGA). В этом случае на кристалл к процессору или SoC добавляется логика FPGA, причем заказчик может заранее указать требуемые соотношения и количество LUT/DSP/BRAM блоков в зависимости от своей специфики, а также добавить свои блоки. Такими вещами занимаются например Menta, Flex-Logic, Achronix.

**Перспективные направления развития вычислительных систем**

• Ещё одна область, в которой проектировщики ищут способы повышения производительности — это память. Традиционно считывание и запись значений всегда были одним из самых серьёзных «узких мест» процессоров. Нам могут помочь быстрые и большие кэши, но считывание из ОЗУ или с SSD может занимать десятки тысяч тактовых циклов. Поэтому инженеры часто рассматривают доступ к памяти как более затратный, чем сами вычисления. Если процессор хочет сложить два числа, то ему сначала нужно вычислить адреса памяти, по которым хранятся числа, выяснить, на каком уровне иерархии памяти есть эти данные, считать данные в регистры, выполнить вычисления, вычислить адрес приёмника и записать значение в нужное место. Для простых инструкций, выполнение которых может занимать один-два цикла, это чрезвычайно неэффективно. Новая идея, которую сейчас активно исследуют — это техника под названием Near Memory Computing. Вместо того, чтобы извлекать небольшие фрагменты данных из памяти и вычислять их быстрым процессором, исследователи переворачивают работу вниз головой. Они экспериментируют с созданием небольших процессоров непосредственно в контроллерах памяти ОЗУ или SSD. Благодаря тому, что вычисления становятся ближе к памяти, существует потенциал огромной экономии энергии и времени, ведь данные теперь не надо передавать так часто. Вычислительные модули имеют прямой доступ к нужным им данным, потому что находятся непосредственно в памяти. Эта идея всё ещё находится в зачаточном состоянии, но результаты выглядят многообещающе.

• Одно из препятствий, которое нужно преодолеть для near memory computing — это ограничения процесса изготовления. Как говорилось, процесс производства кремния очень сложен и в нём задействованы десятки этапов. Эти процессы обычно специализированы для изготовления или быстрых логических элементов, или плотно расположенных накопительных элементов. Если попытаться создать чип памяти с помощью оптимизированного для вычислений процесса изготовления, то получится чип с чрезвычайно низкой плотностью элементов. Если попробовать создать процессор с помощью процесса изготовления накопителей, то получим очень низкую производительность и большие тайминги.

• Одно из потенциальных решений этой проблемы — 3D-интеграция. Традиционные процессоры обладают одним очень широким слоем транзисторов, но это имеет свои ограничения. Как понятно из названия, трёхмерная интеграция — это процесс расположения нескольких слоёв транзисторов друг над другом для повышения плотности и снижения задержек. Вертикальные столбцы, производимые на разных процессах изготовления, могут быть затем использованы для соединений между слоями. Эта идея была предложена уже давно, но индустрия потеряла к ней интерес из-за серьёзных сложностей в её реализации. В последнее время мы наблюдаем возникновение технологии накопителей 3D NAND и возрождение этой области исследований. Кроме физических и архитектурных изменений, на всю полупроводниковую отрасль сильно повлияет ещё одна тенденция — больший упор на безопасность. До недавнего времени о безопасности процессоров думали чуть ли не в последний момент. Это похоже на то, как Интернет, электронная почта и многие другие системы, которые мы сегодня активно используем, разрабатывались почти без оглядки на безопасность. Все существующие меры защиты «прикручивались» по мере случавшихся инцидентов, чтобы мы чувствовали себя в безопасности. В области процессоров такая тактика больно ударила по компаниям, и особенно по Intel. Баги Spectre и Meltdown — это, вероятно, самые известные примеры того, как проектировщики добавляют функции, значительно ускоряющие процессор, не в полной мере осознавая связанные с этим угрозы безопасности. При разработке современных процессоров гораздо большее внимание уделяется безопасности как ключевой части архитектуры. При повышении безопасности часто страдает производительность, но учитывая ущерб, который могут нанести серьёзные баги безопасности, можно с уверенностью сказать, что лучше фокусироваться на безопасности в той же степени, что и на производительности.

• Секвенциальная логика строится аккуратным соединением инверторов и других логических элементов так, чтобы их выходы передавали сигналы обратной связи на вход элементов. Такие контуры обратной связи используются для хранения одного бита данных и называются статическим ОЗУ (Static RAM), или SRAM. Эта память называется статическим ОЗУ в противовес динамической (DRAM), потому что сохраняемые данные всегда напрямую соединены с положительным напряжением или заземлением. Стандартный способ реализации одного бита SRAM — это показанная ниже схема из 6 транзисторов. Самый верхний сигнал, помеченный как WL (Word Line) — это адрес, и когда он включен, то данные, хранящиеся в этой 1-битной ячейке передаются в Bit Line, помеченную как BL. Выход BLB называется Bit Line Bar; это просто инвертированное значение Bit Line. Вы должны узнать два типа транзисторов и понять, что M3 с M1, как и M4 с M2, образуют инвертор.

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

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

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

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

5. Операция умножения в двоичной системе исчисления

6. IEEE754. Специальные числа. Зачем нулю знак.

7. Деление двоичных чисел (общие правила).

8. Деление двоичных чисел с восстановлением остатка.

9. Деление двоичных чисел без восстановления остатка.

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

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

12. Операции с двоичными числами в дополнительном и обратном кодах.

13. Модифицированные коды

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

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

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

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

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

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

20. Неосновные арифметические операции. Вычисление квадратного корня

21. Методы вычисления элементарных функций.

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

23. Погрешности обусловленные форматом с плавающей точкой

24. Основные понятия алгебры логики. Способы задания логической функции.

25. Понятие о принципе двойственности. Суперпозиция логических функций.

26. Нормальная и совершенные нормальные логических функций.

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

28. Минимизация логических выражений методом Квайна.

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

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

31. Законы и правила алгебры Буля

32. Параллелизм. Виды, организация.

33. Устройства ЭВМ. Состав АЛУ.

34. Типы памяти.

35. Код Грея.

36. Обратная польская запись.

37. АЦП и ЦАП. Предназначение. Параметры сравнения и выбора.

38. Корректирующие коды. Код Хэ́мминга. Область применения.

39. Языки описания аппаратуры. ПЛИС (FPGA) модули.

40. Сумматор. Многоразрядный сумматор. Ускорение выполнения математических операций.

41. Полная система логических функций.

42. Искусство управления сложностью. Цифровая абстракция.

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

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

45. Передаточная характеристика. Статическая дисциплина.

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

47. Логический вентиль НЕ и другие на КМОП-транзисторах. Псевдо n-МОП-Логика Потребляемая мощность

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

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

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

51. Проектирование последовательностной логики. Защелки и триггеры. RS-триггер. D-защелка. DТриггер. Регистр.

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

53. Конечные автоматы. Пример проектирования конечного автомата

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

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

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

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

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

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

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

61. Архитектура процессора. Основные компоненты. Способы классификации. Много уровневая организация. Контроллеры вввода-вывола.

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

63. Сравнительная характеристика архитектур. В чем преимущества. Преимущества RISK. Какова проблема лицензирования архитектур.

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

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

66. Что такое суперскалярная архитектура. Ее особенности. Предсказатели переходов. Иерархия памяти. Ветвление 67. Что такое гетерагенные вычисления. FPGA-акселератор? Сфера применения. Перспективные направления развития вычислительных систем.