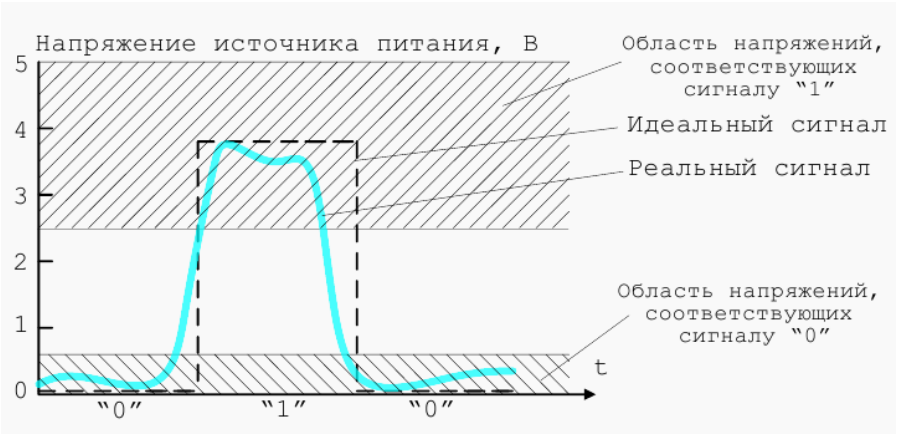
Оглавление

1. Две формы представления информации. Способы представления дискретной информации. Системы счисления, используемые в вычислительной технике: двоичная, 8-я, 10-я, 16-я, двоично-десятичная.

Первая форма представления информации называется аналоговой или непрерывной (с помощью сходной величины – аналога). Количество значений, которые может принимать величина, представленная в такой форме бесконечно велико, даже если величина изменяется в ограниченном диапазоне. Отсюда названия – непрерывная величина и непрерывная информация. Слово непрерывность выделяет основное свойство таких величин – отсутствие разрывов,

промежутков между значениями, которые может принимать аналоговая величина. Величина представляется в виде одного сигнала, пропорционального этой величине. Эта форма представления используется в аналоговых вычислительных машинах



Вторая форма представления информации называется цифровой или дискретной (с помощью набора напряжений, каждое из которых соответствует одной из цифр представляемой величины). Такие величины, принимающие не все возможные, а лишь вполне определённые значения, называются дискретными (прерывистыми). В отличие от непрерывной величины количество значений дискретной величины всегда будет конечным. Величина представляется в виде нескольких сигналов, каждый из которых соответствует одной из цифр заданной величины. Эта форма представления используется в электронных вычислительных машинах (ЭВМ).

Каждое значение из набора исходных данных задачи, результатов её решения может быть представлено в ЭВМ в виде нескольких электрических сигналов, один из которых соответствует числу единиц в значении, другой – числу десятков, третий – числу сотен и т.д. Однако такое представление не является наилучшим с технических позиций. Устройство, предназначенное для обработки подобных сигналов, должно различать в каждом из них десять состояний. Значительно проще построить устройство, которое различало бы всего два состояния (его наличие или отсутствие). Это тем более целесообразно, т.к. существующие сейчас дешёвые устройства для ввода данных в ЭВМ также кодируют отдельные составляющие вводимой информации с помощью двух состояний.

Это натолкнуло создателей первых ЭВМ применение другой системы счисления при внутреннем представлении чисел в машинах: вместо привычной десятичной системы счисления была взята двоичная. 2СС также является позиционной СС, т.е. в ней значение каждой цифры зависит от позиции этой цифры в записи числа.

Существуют специальные термины, широко используемые в вычислительной технике: бит, байт и слово.

Двоичный разряд – бит

Восьмибитовая единица – байт

ЭВМ содержит большое количество ячеек памяти и регистров для хранения двоичной информации. Большинство этих ячеек имеет одинаковую длину n, т.е. они используются для хранения n бит двоичной информации. Информация, хранимая в такой ячейки, называется словом.

Удобная для использования в ЭВМ двоичная система счисления совсем неудобна для записи и чтения чисел человеком. Для сокращения трудоёмкости ручной обработки кодов чисел, команд широко применяют 8- и 16СС. В 8 СС используется 8 цифр (0-7), в 16СС – 10 цифр и 6 прописных букв (0-9, A-F). Т.к. основанием 8СС является $8=2^3$, то для перевода двоичных чисел в

восьмеричные необходимо разделить двоичные числа на триады. Каждую из таких групп можно представить одной восьмеричной цифрой. Аналогичным образом осуществляется перевод двоичных чисел в шестнадцатеричные. Только в этом случае двоичное число разбивается на 4 тетрады, которые представляются одной шестнадцатеричной цифрой.

Наконец следует упомянуть о двоично-десятичной СС, которая используется в цифровых устройствах, где основная часть операций связана не с обработкой и хранением вводимой информации, а с самим её выводом на какие-либо на какие-либо индикаторы с десятичным представлением полученных результатов. В 2-10СС десятичные цифры от 0 до 9 представляют 4-разрядными двоичными комбинациями от 0000 до 1001. Две двоично-десятичные цифры составляют 1 байт (можно представлять значения от 0 до 99)

2. Представление чисел с фиксированной точкой. Прямой, обратный и дополнительный код. Формирование битовых признаков переноса, переполнения, отрицательного результата, нуля.

Целые двоичные числа без знака можно использовать для представления нуля и целых положительных чисел. При размещении таких чисел в одном 16-разрядном слове они могут изменяться от $(0000\ 0000\ 0000\ 0000)_2 = (0000)_{16} = 0$ до $(1111\ 1111\ 1111\ 1111)_2 = (FFFF)_{16} = 2^{16} - 1 = 65535$. Такая запись называется прямым кодом числа.

Подобные числа (так же как и рассмотренные ниже двоичные числа со знаком) относятся к числам с фиксированной запятой, разделяющей целую и дробную части числа. В числах, используемых в базовой ЭВМ, положение запятой строго фиксировано после младшего бита слова.

Целые двоичные числа со знаком используются тогда, когда необходимо различать положительные и отрицательные числа. В современных ЭВМ для представления целых чисел со знаком используется дополнительный код, в котором старший бит формата определяет знак числа: 0 - для положительных чисел и 1 - для отрицательных чисел. При этом дополнительный код положительного числа совпадает с его прямым кодом. А для представления отрицательного числа в дополнительном коде производится инвертирование прямого кода модуля числа (получение обратного кода числа) и добавление к результату единицы. Такая же операция используется при изменении знака числа, представленного в дополнительном коде.

Использование дополнительного кода упрощает конструкцию ЭВМ, так как при сложении двух таких чисел, имеющих разные знаки, не требуется переходить к операциям вычитания меньшего (по модулю) числа из большего и присвоения результату знака большего числа. Кроме того, одной и той же схемой сумматора можно воспользоваться для выполнения операций над знаковым и беззнаковым представлением числа.

Признаком выхода за границы разрядной сетки для беззнакового представления числа является перенос в старший разряд (бит C - Carry). Например, при сложении:

$$\begin{array}{r} 1000\ 0000\ 0000\ 0000 \\ +\ 1000\ 0000\ 0000\ 0000 \\ \hline 1\ 0000\ 0000\ 0000\ 0000 \end{array}$$

В ответе должно получиться 32768+32768=65536, но т.к. разрядность слова составляет лишь 16 бит, то в нем сохраняется только часть результата, т.е. 0. Единица, возникшая вследствие переноса оказалась в несуществующем 17 разряде.

Признаком переполнения разрядной сетки для знакового представления

является бит переполнения (OVerflow). Разные знаки слагаемых, или совпадение знаков слагаемых со знаком суммы свидетельствуют о том что результат корректен. в противном случае формируется сигнал – Переполнение

Признак отрицательного результата N при знаковом представлении выставляется в случае когда в старшем разряде числа в доп. коде находится 1

Признак нулевого результата Z выставляется в случае когда все разряды числа равны 0

3. Представление символьных и строковых данных. Принципы построения кодовых таблиц ASCII, КОИ-8, ISO8859-5, Windows-1251, UTF-8, UTF-16.

Представление текстовой информации в вычислительных машинах основано на кодировании букв алфавитов для существующих на земле языков, которые традиционно используются человечеством. Символ - это графическое изображение, которое используется человеком для создания слов, текстов и другой значимой информации. Как известно, к символам относятся буквы, знаки препинания, символы валют, цифры и т.д.

В ЭВМ представление текстовой информации основывается на кодировании букв и символов при помощи кодовой таблицы. Кодовая таблица (или кодировка символов) является соглашением между разработчиками о соответствии каждому символу определенного порядкового номера или кода, чтобы его можно было сохранять в памяти ЭВМ или передавать по каналам связи.

Для хранения графических начертаний символов в ЭВМ существуют *шрифты.* Шрифты бывают векторные и растровые. В растровых шрифтах каждому коду символа соответствует изображение, определенного (в точках) размера. В векторных хранится принцип начертания символа в виде последовательности линий.

Кодировки стандарта ASCII

ASCII — таблицы кодировок, в которых содержатся основные символы (английский алфавит, цифры, знаки препинания, символы национальных алфавитов(свои для каждого региона), служебные символы) и длина кода каждого символа n = 8 бит.

7 бит:

ASCII — первая кодировка, пригодная для работы с текстом. Помимо маленьких букв английского алфавита и служебных символов, содержит большие буквы английского языка, цифры, знаки препинания и другие символы (при этом старший бит использовался для контроля четности битов передаваемого по каналом связи символа).

Кодировки стандарта ASCII (8 бит):

ISO 8859 — первая кодировка, в которой стало возможно использовать символы национальных алфавитов.

КОИ8-R — первая русская кодировка. Символы кириллицы расположены не в алфавитном порядке. Их разместили в верхнюю половину таблицы так, чтобы позиции кириллических символов соответствовали их фонетическим аналогам в английском алфавите. Это значит, что даже при потере старшего бита каждого символа, например, при проходе через устаревший семибитный модем, текст остается "читаемым" (отсюда и появилось понятие транслита).

Windows-1251 — русская кодировка, использовавшаяся в русскоязычных версиях операционной системы Windows в начале 90-х годов. Кириллические символы идут в алфавитном порядке. Содержит все символы, встречающиеся в типографике обычного текста (кроме знака ударения).

Кодировки стандарта UNICODE

Юникод — это промышленный стандарт, обеспечивающий цифровое представление символов всех письменностей мира и специальных символов.

Стандарт предложен в 1991 году некоммерческой организацией «Консорциум Юникода» (англ. Unicode Consortium). Применение этого стандарта позволяет закодировать очень большое число символов из разных письменностей. Стандарт состоит из двух основных разделов: универсальный набор символов (англ. UCS, universal character set) и семейство кодировок (англ. UTF, Unicode transformation format). Универсальный набор символов задаёт однозначное соответствие символов кодам — элементам кодового пространства, представляющим неотрицательные целые числа. Семейство кодировок определяет машинное представление последовательности кодов UCS.

Способы представления

Юникод имеет несколько форм представления (англ. Unicode Transformation Format): UTF-8, UTF-16 (UTF-16BE, UTF-16LE) и UTF-32 (UTF-32BE, UTF-32LE).

UTF-8

UTF-8 — представление Юникода, обеспечивающее наилучшую совместимость со старыми системами, использовавшими 8-битные символы. Текст, состоящий только из символов с номером меньше 128, при записи в UTF-8 превращается в обычный текст ASCII. И наоборот, в тексте UTF-8 любой байт со значением меньше 128 изображает символ ASCII с тем же кодом. Остальные символы Юникода изображаются последовательностями длиной от двух до шести байт.

UTF-16

Первая версия Юникода (1991 г.) представляла собой 16-битную кодировку с фиксированной шириной символа; общее число разных символов было 65 536. Во второй версии Юникода (1996 г.) было решено значительно расширить кодовую область; для сохранения совместимости с теми системами, где уже был реализован 16-битный Юникод, и была создана UTF-16. Область 0xD800—0xDFFF, отведённая для суррогатных пар, ранее принадлежала к области «символов для частного использования».

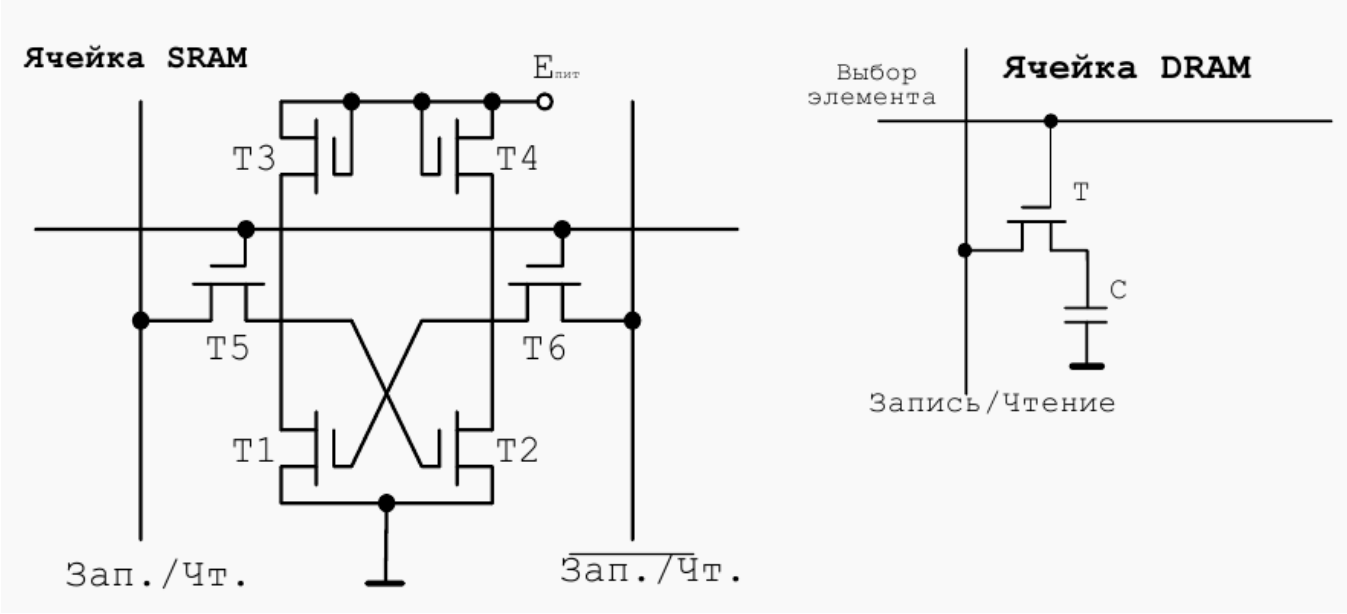
Поскольку в UTF-16 можно отобразить 1 112 064 символов, то это число и было выбрано в качестве новой величины кодового пространства Юникода.

4. Базовые элементы вычислительной техники: ячейки, регистры, шины, вентили, тактовые генераторы, логические схемы, триггеры, регистры, счетчики, сумматоры.

Ячейка памяти – минимальный адресуемый элемент запоминающего устройства ЭВМ. Ячейки имеют адрес (порядковый номер, число), по которому к ним могут обращаться команды процессора. Ячейки памяти состоят из элементов, которые могут находиться в одном из двух устойчивых состояний: конденсатор заряжен или разряжен, транзистор находится в проводящем

или непроводящем состоянии. Одно из таких физических состояний создает высокий уровень выходного напряжения элемента памяти, а другое – низкий. Первое обычно принимается за двоичную 1, а второе – за двоичный 0. Возможно и обратное кодирование. Хотя переход от 0 к 1 и от 1 к 0 происходит не мгновенно, однако в определенные моменты времени этот сигнал достигает значений, которые воспринимаются элементами ЭВМ как 0 или 1.

Память бывает статическая (SRAM - *static random access memory*) и динамическая (DRAM – dynamic ...)



Регистр процессора – память внутри процессора, предназначенная для хранения адресов и промежуточных результатов вычислений или данных, необходимых для работы самого процессора. Регистр характеризуется единственным числом: количеством битов, которые могут в нем храниться. Операция чтения информации, хранимой в регистре, сводится к созданию копии его содержимого, оригинал же сохраняется в регистре без изменений.

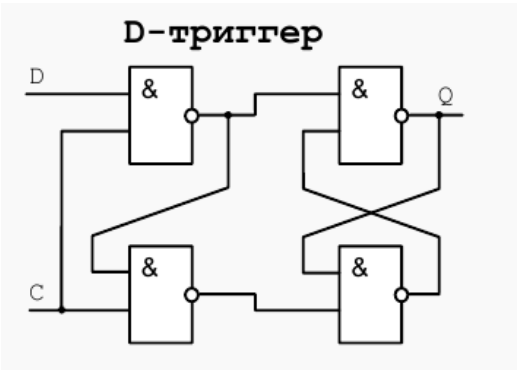
Шина - электрическая цепь, соединяющая регистр с другим регистром или иным устройством ЭВМ. Шина состоит из параллельных проводов, каждый из которых предназначен для передачи соответствующего регистра. Также шина содержит несколько дополнительных проводов, используемых для передачи сигналов синхронизации и управления. Шины служат для передачи информации лишь в направлении, обозначенном стрелкой на шине. Специальные схемы позволяют в одни моменты времени передавать информацию по шине в одну сторону, а в другие – в обратном направлении, т.е. организовать двунаправленную шину.

Вентильные схемы – это электронные ключевые схемы, предназначенные для управления потоком информации из регистров в шины и обратно. Такая схема содержит два входа и один выход. На один вход подается информационный сигнал (данные с регистра), а на другой (являющийся вентилем) – управляющий. Если управляющий сигнал равен 1, то данные проходят схему без препятствий, если 0 – никакая информация не пройдет через схему. Для подачи информационного сигнала на вход вентильной схемы обычно используется многопроводная шина. Для передачи выходного сигнала требуется шина с таким же количеством проводов. Если управляющий сигнал равен 1, то информационные сигналы на входной и выходной шинах совпадают.

Тактовый генератор – устройство, генерирующее электрические импульсы заданной частоты (обычно прямоугольной формы). Используется для синхронизации процессов передачи информации между устройствами ЭВМ.

Функциональная логическая схема - совокупность логических элементов (простейшее устройство ЭВМ, выполняющее одну определённую логическую операцию над входными сигналами согласно правилам алгебры логики) и связей между ними.

Триггер — класс электронных устройств, обладающих способностью длительно находиться в одном из двух устойчивых состояний и чередовать их под воздействием внешних сигналов. Каждое состояние триггера легко распознаётся по значению выходного напряжения. Отличительной особенностью триггера как функционального устройства является свойство запоминания двоичной информации. Под памятью триггера подразумевают способность оставаться в одном из двух состояний и после прекращения действия переключающего сигнала.

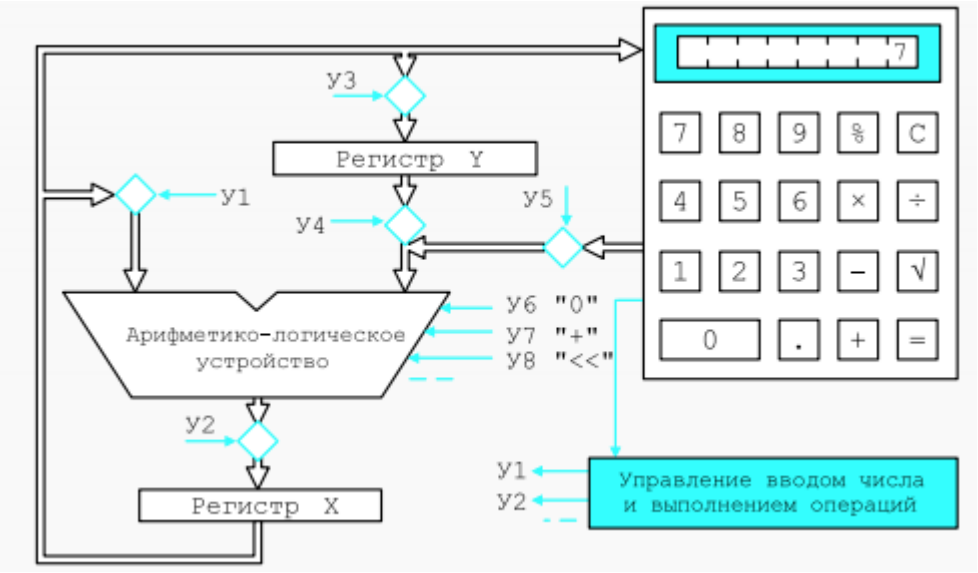


Счётчик числа импульсов — устройство, на выходах которого получается двоичный (двоично–десятичный) код, определяемый числом поступивших импульсов. Основной параметр счётчика —модуль счёта —максимальное число единичных сигналов, которое может быть сосчитано счётчиком.

Сумматор —устройство, преобразующее информационные сигналы (аналоговые или цифровые) в сигнал, эквивалентный сумме этих сигналов.

5. Структура и принцип функционирования ЭВМ. Порядок функционирования простого процессора на примере калькулятора.

Типичная ЭВМ состоит из процессора, памяти и устройств ввода-вывода.
«Сердцем» ЭВМ является процессор, в состав которого входят устройство управления выборкой команд из памяти и их выполнением, арифметико-логическое устройство, производящее операции над данными, регистры, осуществляющие временное хранение данных и состояний процессора, схемы для управления и связи с подсистемами памяти и ввода-вывода.
Устройство ввода обеспечивает считывание информации с определенных носителей информации и ее представление в форме электрических сигналов, воспринимаемых другими устройствами ЭВМ. Устройства вывода представляют результаты обработки информации в форме, удобной для визуального восприятия. Память ЭВМ включает устройство, обеспечивающее хранение команд и данных. Это устройство состоит из блоков одинакового размера – ячеек памяти, предназначенных для хранения одного слова информации.
Ячейка памяти состоит из элементов памяти, состояние каждого из которых соответствует одной двоичной цифре. Совокупность нулей и единиц, хранящихся в элементах одной ячейки, представляет собой содержимое этой ячейки памяти. В микро ЭВМ используются безадресные, одноадресные и реже двухадресные команды. В одноадресных командах один из операндов выбирается из специального регистра – аккумулятора. В него же заносится и результат операции. Безадресные команды или задают какое-либо действие с устройствами ЭВМ, или используются для работы с операндами, имеющими фиксированное расположение (чаще всего с аккумулятором). В процессе работы ЭВМ последовательно выполняет набор достаточно простых операций: выборку команды, определение ее типа, исполнение команды и определение адреса следующей команды.



Рассмотрим принципы функционирования простейшей ЭВМ — калькулятора. Он состоит из двух регистров — X и У, хранящих результаты ввода пользователя и промежуточных вычислений, АЛУ, которая может выполнять простейшие арифметические и логические операции, шин и управляющих вентилей, осуществляющих передачу данных между функциональными блоками калькулятора, устройства управления (УУ), клавиатуры и дисплея.
Дисплей постоянно отображает содержимое регистра X (проследите о шине путь информации от X к дисплею, убедитесь, что вентили на этом пути отсутствуют). Клавиатура передает значение нажатой клавиши на вентиль У5, каждое нажатие на клавишу запускает УУ, которое в зависимости от текущего состояния ЭВМ формирует последовательность импульсов для выполнения требуемой операции, которая называется *циклом* импульсов. Каждая группа импульсов выдается последовательно, в моменты, совпадающие с импульсами тактового генератора.
Предположим пользователь вводит первую цифру необходимого ему числа (7). Так как это новая операция, УУ, после своей активации нажатием кнопки 7, выдаст последовательность управляющих импульсов для первой цифры числа.
В первую очередь необходимо сохранить предыдущее значение регистра X в регистре У. Для этого должен быть открыт вентиль, управляющий записью в регистр У. Он открывается управляющим сигналом У3.
После этого необходимо обнулить регистр X, подготовив его для новой цифры числа, которая была введена с клавиатуры. Для этого должны быть закрыты все вентили, кроме У2 — который осуществляет передачу данных из АЛУ в регистр У и У6 — который сформирует в АЛУ значение 0.
Далее необходимо сложить значение 0 с цифрой с клавиатуры. Для этого содержимое регистра X поступает на правый вход АЛУ (У1), цифра с клавиатуры на правый вход АЛУ (У5), и выбирается операция сложения (У7).
В конце цикла необходимо передать результат сложения в регистр X (У2), отобразив его, при этом, на дисплее.

Для ввода второй и последующих цифр необходимо осуществлять поразрядный сдвиг регистра X после каждой введенной цифры и складывать введенную цифру со сдвинутым содержимым регистра X. Разберем этот цикл по тактам:

1. Содержимое регистра X через вентиль, управляемый У1 подается на левый вход АЛУ, при этом вентили У4, который управляет передачей из регистра У и У5 (ввод с клавиатуры), должны быть закрыты, при этом на правый вход АЛУ подается 0. Управляющий сигнал У8 вызовет сдвиг информации, которая поступает на левый вход АЛУ. В двоично-десятичной системе счисления (в которой обычно выполняют вычисления калькуляторы) сдвигу на один десятичный разряд соответствует умножение на 10.
2. Результат сдвига записывается (управляющий сигнал У2) в регистр X.

3. Затем необходимо сложить результат сдвига с новой цифрой с клавиатуры. Для этого содержимое регистра X поступает на правый вход АЛУ (У1), цифра с клавиатуры на правый вход АЛУ (У5), и выбирается операция сложения (У7).
4. Результат сдвига и сложения записывается (управляющий сигнал У2) в регистр X.

Когда пользователь нажимает кнопку «+» или «=», в зависимости является ли это сложение промежуточной или конечной операцией, необходимо сложить содержимое регистра X и регистра У. Для этого:
5. Содержимое регистра X подается на левый вход АЛУ (У1), содержимое регистра У подается на правый вход АЛУ (У4) и выполняется операция сложения (У7). Все остальные вентили при этом закрыты (в первую очередь должен быть закрыт У5).
6. Результат записывается в регистр X и показывается на экране калькулятора.

6. Операционная система Unix — ядро ОС и файловая система.

From wiki: https://ru.wikipedia.org/wiki/UNIX

UNIX — семейство переносимых, многозадачных и многопользовательских операционных систем. Идеи, заложенные в основу UNIX, оказали огромное влияние на развитие компьютерных операционных систем. В настоящее время UNIX-системы признаны одними из самых исторически важных ОС.

Основное отличие UNIX-подобных систем от других операционных систем заключается в том, что это изначально многопользовательские многозадачные системы. То есть в один и тот же момент времени сразу множество людей может выполнять множество вычислительных задач (процессов). Даже популярную во всём мире систему Microsoft Windows нельзя назвать полноценной многопользовательской системой, так как кроме как на некоторых серверных версиях, в один и тот же момент за одним компьютером с Windows может работать только один человек. В Unix может работать сразу много людей, при этом каждый из них может выполнять множество различных вычислительных процессов, которые будут использовать ресурсы именно этого компьютера.

Вторая колоссальная заслуга Unix в её мультиплатформенности. Ядро системы написано таким образом, что его легко можно приспособить практически под любой микропроцессор.

UNIX имеет и другие характерные особенности:

- использование простых текстовых файлов для настройки и управления системой;
- широкое применение утилит, запускаемых из командной строки;
- взаимодействие с пользователем посредством виртуального устройства — терминала;
- представление физических и виртуальных устройств и некоторых средств межпроцессового взаимодействия в виде файлов;
- использование конвейеров из нескольких программ, каждая из которых выполняет одну задачу.

Файловая система UNIX

From http://works.doklad.ru/view/r8f_Kyf1Whs.html

В операционной системе UNIX файл является хранилищем двоичных и символьных данных, хранимых как поток байтов.

Понятие файла является одним из наиболее важных для ОС UNIX. Все файлы, с которыми могут манипулировать пользователи, располагаются в файловой системе, представляющей собой дерево, промежуточные вершины которого соответствуют каталогам, и листья - файлам и пустым каталогам.Каждый каталог и файл файловой системы имеет уникальное полный путь. Каталог, являющийся корнем файловой системы (корневой каталог) имеет путь /. Коротким или относительным путем называется путь к файлу от текущего рабочего каталога.В каждом каталоге содержатся два специальных файла-ссылки, файл "." - ссылка на текущий каталог, и ссылка ".." на родительский каталог.

inode - Index-node - описатель файла, его уникальный номер. Он содержит всю информацию о файле, за исключением имени файла, и собственно данных файла. В inod'е хранится:
тип файла, права, время модификации/создания файла и другая служебная информация под общим названием «метаданные».

7. Операционная система Unix — интерпретаторы, стандартные потоки ввода вывода, фильтры.

Командный интерпретатор – программа, предоставляющая пользователю интерфейс для общения с командной строкой; эта программа «переводит» введенные пользователем команды на понятный операционной системе язык. Интерпретатор более известен как ***оболочка*** *(англ. shell)*. Наиболее распространенными оболочками являются *sh, bash* (стандарт в Unix)*, c shell.* Пользователь может вводить команды как по отдельности, так и с помощью набора команд (скриптов). Команды могут задаваться как напрямую в командной строке, так и поступать из *стандартного ввода* или указанного файла. В качестве команд могут приниматься вызовы системных или прикладных утилит или управляющие конструкции. Кроме того, оболочка отвечает за перенаправление потоков ввода-вывода. В совокупности с набором утилит, она представляет собой операционную среду, язык программирования и средства решения как системных, так и прикладных задач, особенно по части автоматизации выполняемых последовательностей команд.
Для взаимодействия и обмена информацией с пользователем используются файлы, именуемые **стандартными потоками ввода** (для чтения из него) и **вывода** (для записи в него). Вывод на экран представляется тоже как запись в файл, а ввод – как чтение из файла. Кроме потоков ввода и вывода существует так же **стандартный поток ошибок**, на который выводится вся служебная информация, которая не должна попадать в поток вывода (сообщения об ошибке или ходе работы программы).
Стандартные потоки привязаны к *файловым дескрипторам* с номерами:
0 для ввода (stdin) 1 для вывода (stdout) 2 для ошибок (stderr).

Потоки по умолчанию связаны с терминалом (командной строкой), но их можно подключить к чему угодно – к файлам, программам или устройствам. В интерпретаторе такая операция называется *перенаправлением*. Таким образом, стандартные потоки можно перенаправлять не только в файлы, но и на вход других программ.
Для осуществления перенаправления используются следующие операции:

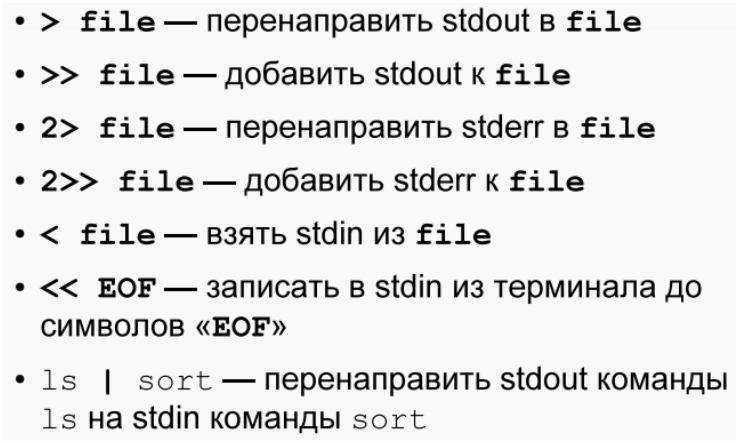
Команда > файл (или >>)
Выполняется команда, а вывод помещается в файл (или добавляется в конец).

Команда < файл
Файл используется в качестве источника ввода. При этом на каждый запрос ввода программы считывается 1 строка текста из файла.

Команда1 | команда2
*Вывод команды1 пойдет в качестве ввода на команду2 без использования промежуточных файлов. Такая возможность называется **конвейером.***

Команда 2> файл
Поток ошибок направляется в файл. По умолчанию этот поток выводится на стандартный вывод.

Команда 2>&1 файл (или &> или >&)
Такой синтаксис используется для объединения потоков вывода и потока ошибок для обработки их вместе.



Файл т.н. «пустое устройство» - /dev/null – перенаправление в него позволяет избавиться от ненужных сообщений об ошибке или игнорирования вывода. С помощью него также можно создавать пустые файлы, используя в качестве источника ввода. При записи в него может вместить любое количество информации, он работает в качестве «черной дыры».

Фильтры:

wc, grep, sort

8. Операционная система Unix — основные команды, права файлов и способы их задания.

touch файл
Создает пустой файл, а если он уже есть – обновляет время последней модификации.

mkdir каталог
Создает пустой каталог.

rm файл
Удаляет файл.
–r
Рекурсивно стирает каталоги. Если этого флага нет, файл не может быть каталогом.

rmdir каталог
Стирает только пустые каталоги.

echo
Выводит строку текста.

cat файл
Выводит содержимое файла.

pwd
Выводит имя текущего каталога.

ls файл
Выводит список файлов в каталоге или информацию о файле, если это не каталог.
–l
Длинный формат. Выводится с подробной информацией о каждом файле.
–a
Вывод вместе со скрытыми файлами.
–F
К имени файла добавляется его тип.
–R
Рекурсивно выводит подкаталоги.

cd каталог
Переходит в каталог.

cp файл1 файл2
Копирует файл в другой файл.

mv файл каталог
Перемещает файл в каталог.

ln файл1 файл2
Создает новую жесткую ссылку на файл. Жесткая ссылка может ссылаться только в пределах одного диска. Файл не будет удален, пока на него есть хоть одна жесткая ссылка.
–s
Создает символическую ссылку. Может ссылаться куда угодно. Если переместить/удалить файл, симв. ссылка будет недействительна.

head/tail файл
Выводит первые/последние 4 строки файла
–n
Первые/последние n строк.
–c
Первые/последние c байт.

wc файл
Выводит количество строк, слов и байт в файле.
–l
Только кол-во строк.
–w
Только кол-во слов.
–c
Только кол-во байт.
–m
Кол-во символов.

find выражение
Ищет файлы в иерархии каталогов по заданным параметрам.

man команда
Выводит справку по команде.

Права доступа к файлам

Для каждого файла существуют следующие категории пользователей:

u (user)
Владелец файла.
g (group)
Члены группы, владеющей файлом.
o (others)
Все остальные.
a (all)
Все категории. Не рассматривается как отдельная категория.

Каждая из этих категорий может иметь любую комбинацию из следующих прав:

r (read)
Право на чтение файла/просмотр каталога.
w (write)
Право на запись в файл/добавление или удаление каталога.
x (execute)
Право на исполнение файла/поиск и переход в каталог.

Права представляют собой последовательность из 9 бит – по 3 бита на категорию: владелец, группа, прочие; в следующем порядке – чтение, запись, исполнение. В случае отсутствия какого-либо из прав у категории, ставится символ «-».
Вторым способом записи прав является запись этой последовательности в 8-ричной системе счисления, где праву на чтение (r) соответствует цифра 4, праву на запись (w) – цифра 2, а праву на исполнение (x) – цифра 1. Цифра 0 означает отсутствие прав. Для получения конечной цифры, нужные права суммируются. Таким образом, запись занимает всего 3 бита: по 1 биту на категорию.

Для выставления прав файлу (каталогу) используется команда chmod.
Существует 3 способа задания прав доступа:

chmod [ugoa]{+-=}[rwx] файл
Добавляет, удаляет или устанавливает выбранную комбинацию прав для выбранной комбинации категорий.
chmod число файл
Устанавливает права на основе восьмиричной записи.
chmod категория1=категория2 файл
Копирует права одной категории и присваивает их другой.

Сначала рассмотрим какими бывают права доступа linux и как они устанавливаются. Пред этим рекомендую прочитать статью про права, ссылка на которую есть выше. Есть три основных вида прав:

- **r** - чтение;
- **w** - запись;
- **x** - выполнение;
- **s** - выполнение от имени суперпользователя (дополнительный);

Также есть три категории пользователей, для которых вы можете установить эти права на файл linux:

- **u** - владелец файла;
- **g** - группа файла;

- **o** - все остальные пользователи;

Синтаксис настройки прав такой:

группа_пользователейдействиевид_прав

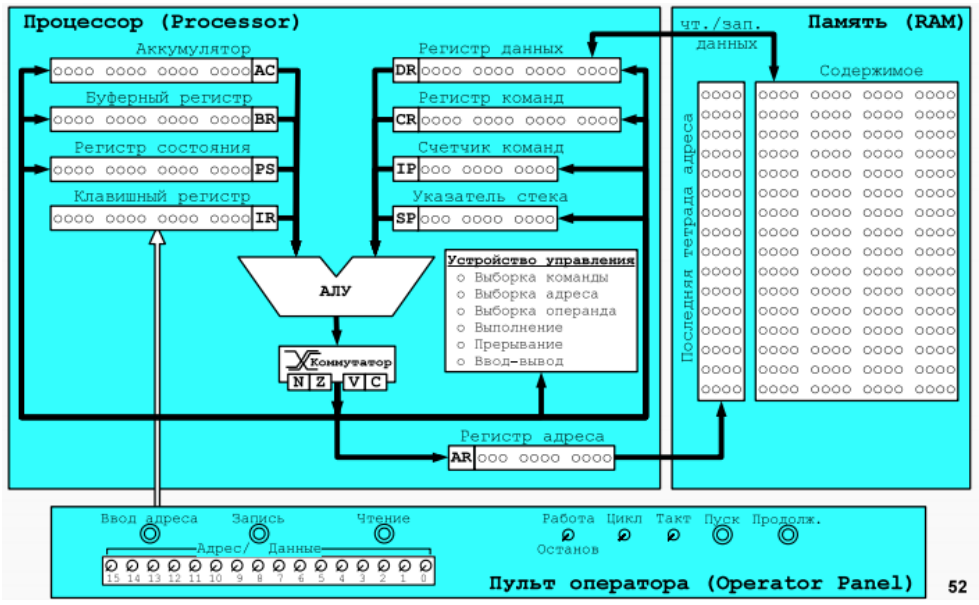
В качестве действий могут использоваться знаки "**+**" - включить или "**-**" - отключить. Рассмотрим несколько примеров:

- **u+x** - разрешить выполнение для владельца;
- **ugo+x** - разрешить выполнение для всех;
- **ug+w** - разрешить запись для владельца и группы;
- **o-x** - запретить выполнение для остальных пользователей;
- **ugo+rwx** - разрешить все для всех;

Но права можно записывать не только таким способом. Есть еще восьмеричный формат записи, он более сложен для понимания, но пишется короче и проще. Я не буду рассказывать, как считать эти цифры, просто запомните какая цифра за что отвечает, так проще:

- **0** - никаких прав;
- **1** - только выполнение;
- **2** - только запись;
- **3** - выполнение и запись;
- **4** - только чтение;
- **5** - чтение и выполнение;
- **6** - чтение и запись;
- **7** - чтение запись и выполнение.

9. Состав и структура БЭВМ. Адресные пространства БЭВМ. Система команд БЭВМ, форматы команд. Машинные циклы.



БЭВМ включает в себя нескольких функциональных блоков и регистров:

- Память - состоит из 2048 ячеек. Каждая ячейка занимает 16 разрядов. Для обращения к памяти существует два регистра: 11-разрядный *регистр адреса* (AR – Address Register), в который нужно поместить адрес прежде чем обратиться к памяти; 16-разрядный *регистр данных* (DR – Data Register), который предназначен для чтения или записи данных в/из ячеек памяти. Чтение данных и запись данных реализуется по шинам, которые подключаются к ячейке памяти.
- 11-разрядный *счетчик команд* (IP – Instruction Pointer). Хранит в себе адрес следующей исполняемой команды.
- *Арифметико-логическое устройство* или *АЛУ* (ALU – Arithmetic-n-Logic Unit) может выполнять несколько операций: сложение, логическое умножение, инверсия и прибавление единицы. При операций «сложение» возможен выход за пределы разрядной сетки и формирование битов переполнения и переноса. Выход из АЛУ через коммутатор подключается к шине, по которой информация может быть передана в любой другой регистр БЭВМ.
- *Буферный регистр* (BR – Buffer Register) это 16-разрядный регистр, который используется для организации промежуточного хранения данных во время работы.
- *Регистр команд* (CR – Command Register) - используется для хранения кода команды и декодирования операций, происходящих во время работы.
- *Аккумулятор* (AC - ACcumulator). БЭВМ относится к ЭВМ, которые называются ЭВМ аккумуляторного типа, где все вычисления с данными производятся через этот регистр.
- *Указатель стека* (SP – Stack Pointer), как и ЕР и АЯ 11-ти разрядный, и всегда указывает на вершину стека - особого участка памяти, который предназначен для хранения адресов возвратов и параметров подпрограмм и прерываний.
- 16-разрядный *клавишный регистр* (IR - Input Register) - находится в составе *пульта оператора* ЭВМ и предназначен для ввода адреса программы, кодов программы и данных, запуска программы на выполнение и управления режимами работы БЭВМ.
- 16-ти разрядный *регистр состояния* (PS – Program State) хранит биты управляющие работой БЭВМ (работа, прерывание и пр.) и признаки результата.

Устройство управления разработано в виде *микропрограммного устройства управления* (МПУ, MCU – Microprogram Control Unit) — простейшего компьютера, программа которого непосредственно состоит из *микроопераций* - т. е. по-тактного изменения значений вентилей БЭВМ, которые задают атомарные операции: вычисления в АЛУ, пересылки данных между регистрами и простейшие проверки. Код программы для МПУ называется *микрокодом.*

МПУ выполняет все машинные команды БЭВМ. Исполнение в МПУ машинной команды называется *циклом команды.* Цикл команды логически разбит на пять циклов:

- *Цикл выборки команды.* Осуществляет загрузку исполняемой команды в регистр команды и частичное ее декодирование. Выполнятся для каждой исполняемой команды.
- *Цикл выборки адреса.* Предназначен для обработки адресных команд и выборки адреса операнда с учетом режимов адресации.
- *Цикл выборки операнда.* Для тех команд, где это необходимо, размещает в БЯ второй операнд команды. Первым, напомним, является аккумулятор.
- *Цикл исполнения.* Производится исполнение команды.
- *Цикл прерывания.* Цикл выполняется в том случае, если разрешены прерывания и устройство ввода-вывода готово к обмену (то есть, требует прерывания — будет обсуждено позднее).

Для обеспечения работы оператора БЭВМ в ней предусмотрена микропрограммная реализация *циклов пультовых операций*:

- *Ввод адреса* — адрес из клавишного регистра вводится в счетчик команд.

- *Чтение* — информация по адресу в 1Р читается из памяти в БЯ, 1Р увеличивается на единицу.
- *Запись* — информация из 1Я записывается в память по адресу в ЕР, 1Р увеличивается на единицу. Используется для ввода программы и данных в режиме оператора.
- *Пуск* — осуществляет сброс состояния БЭВМ и переход к выполнению программы.

На панели оператора, кроме того, расположены другие органы управления — переключатель «Работа/Останов», который вызывает останов программы после каждой команды; переключатель «Такт», который может выполнить микрокод по одному такту, кнопка «Продолжение» - возобновляющая работу остановленной БЭВМ.

10.Организация вычислений в БЭВМ. Сдвиги, арифметические и логические операции. Цикл выборки команды.

Целые двоичные числа без знака можно использовать для представления нуля и целых положительных чисел. В 16-разрядном слове они могут изменяться от 0 до 65535. Это числа с фиксированной запятой. Целые двоичные числа со знаком используются, когда необходимо различать положительные и отрицательные числа. Отрицательные числа представляются в дополнительном коде. Это упрощает конструкцию ЭВМ. Сложение целых двоичных чисел со знаком и без знака выполняется в базовой ЭВМ с помощью команды ADD. По команде INC к содержимому аккумулятора прибавляется единица, а по команде DEC – единица вычитается. Если при этом возникает перенос из старшего разряда А, то в регистр переноса заносится 1, в противном случае в него заносится 0. Вычитание может выполняться путем сложения уменьшаемого и дополнительного кода вычитаемого. В базовой ЭВМ нет команд для выполнения умножения и деления (АЛУ не выполняет таких операций), поэтому произведение и частное необходимо получать программным путем. Для изменения знака числа необходимо его инвертировать, а затем прибавить единицу к младшему разряду (NEG). Побитовая обработка данных обеспечивается командами логического умножения, циклических сдвигов, а также командами инвертирования и очистки регистра переноса.

Команда AND выполняет над каждым разрядом аккумулятора и содержимым ячейки булеву операцию «И». Результат выполнения команды для каждой пары битов операндов равен 1 только тогда, когда оба бита равны 1, а в остальных случаях бит результата равен 0, т.е. команда позволяет выделять или очищать определенные биты слова.

Команды ROL и ROR замыкают аккумулятор и регистр переноса в кольцо и сдвигают все биты кольца влево или вправо. Арифметическими сдвигами числа (ASL и ASR) можно реализовать операции умножения или деления на 2 (один сдвиг), 4 (два сдвига), 8 (три сдвига) и т.д.

Цикл выборки команды:

IP ? BR, AR

BR + 1 ? IP; MEM(AR) ? DR

DR ? CR

if CR(15) = 1 then GOTO CHKBR @ 09

if CR(14) = 1 then GOTO CHKABS @ 0C

if CR(13) = 1 then GOTO CHKABS @ 0C

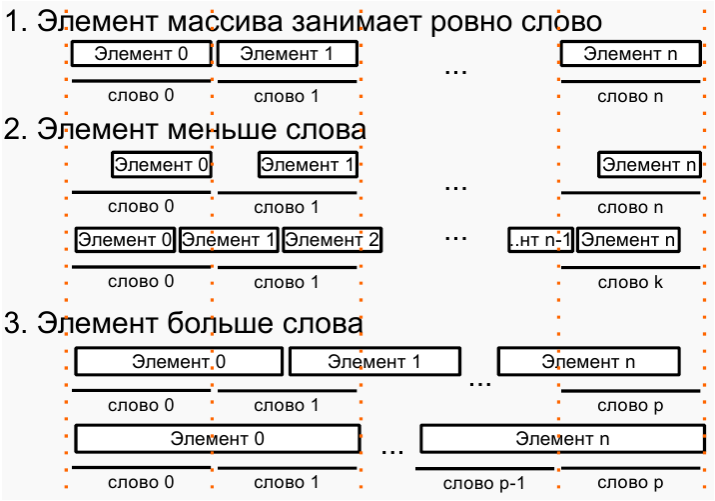
if CR(12) = 0 then GOTO ADDRLESS @ 78

GOTO IO @ C2

11.Организация массивов данных. Режимы адресации. Цикл выборки адреса и операнда БЭВМ.

Код				Мнемоника	Описание	Реализация машинных циклов AF, *OF*
11	10	9	8			
0	M	M	M	**ADD 0ADDR** **ADD $L**	Прямая абсолютная	**CR → DR,** *DR → AR; MEM(AR) → DR*
1	0	0	0	**ADD (L)**	Косвенная относительная	**SXT_CR(0..7) → BR,** **BR + IP → AR, MEM(AR) → DR,** *DR → AR; MEM(AR) → DR*
1	0	0	1		Резерв	
1	0	1	0	**ADD (L)+**	Косвенная автоинкрементная (постинкремент)	**SXT_CR(0..7) → BR,** **BR + IP → AR, MEM(AR) → DR, DR + 1 → DR,** **DR → MEM(AR), DR - 1 → DR,** *DR → AR; MEM(AR) → DR*
1	0	1	1	**ADD -(L)**	Косвенная автодекрементная (предекремент)	**SXT_CR(0..7) → BR,** **BR + IP → AR, MEM(AR) → DR, DR - 1 → DR,** **DR → MEM(AR),** *DR → AR; MEM(AR) → DR*
1	1	0	0	**ADD &N** **ADD (SP+N)**	Косвенная относительная, со смещением (SP)	**SXT_CR(0, 7) → BR,** **BR + SP → DR,** *DR → AR; MEM(AR) → DR*
1	1	0	1		Резерв	
1	1	1	0	**ADD L** **ADD (IP+N)**	Прямая относительная	**SXT_CR(0..7) → BR,** **BR + IP → DR,** *DR → AR; MEM(AR) → DR*
1	1	1	1	**ADD #N**	Прямая загрузка	CR(0, 7) → BR, BR → DR

Где здесь AF и OF?



12. Управление вычислительным процессом в БЭВМ. Команды ветвлений, цикл исполнения команды LOOP.

Задача управления вычислительным процессом, т.е. требуемой последовательностью выполнения команд, решается в базовой ЭВМ при помощи команд перехода, команды «Приращение и пропуск» (LOOP) и «Останов» (HLT).

Цикл исполнения LOOP:

~0 + DR ? DR

~0 + DR ? BR; DR ? MEM(AR)

if BR(15) = 0 then GOTO INT @ C4

IP + 1 ? IP

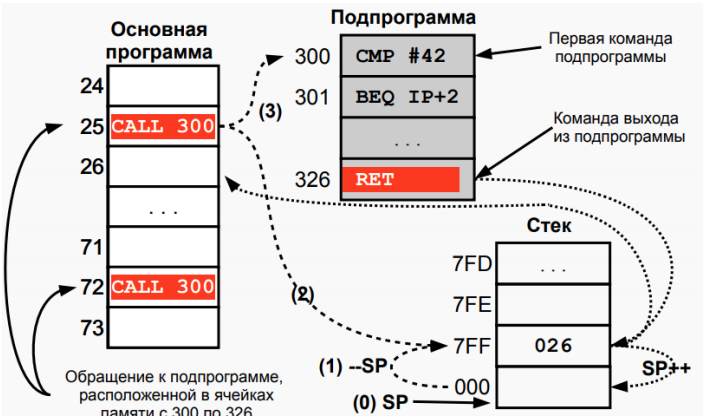
GOTO INT @ C4

Наименование	Мнемон.	Код	Описание
Переход, если равенство	**BEQ D**	F0XX	IF Z==1 THEN IP+D+1 → IP
Переход, если неравенство	**BNE D**	F1XX	IF Z==0 THEN IP+D+1 → IP
Переход, если минус	**BMI D**	F2XX	IF N==1 THEN IP+D+1 → IP
Переход, если плюс	**BPL D**	F3XX	IF N==0 THEN IP+D+1 → IP
Переход, если ниже/перенос	**BLO D** **BCS D**	F4XX	IF C==1 THEN IP+D+1 → IP
Переход, если выше/нет переноса	**BHIS D** **BCC D**	F5XX	IF C==0 THEN IP+D+1 → IP
Переход, если переполнение	**BVS D**	F6XX	IF V==1 THEN IP+D+1 → IP
Переход, если нет переполнения	**BVC D**	F7XX	IF V==0 THEN IP+D+1 → IP
Переход, если меньше	**BLT D**	F8XX	IF N⊕V==1 THEN IP+D+1 → IP
Переход, если больше или равно	**BGE D**	F9XX	IF N⊕V==0 THEN IP+D+1 → IP
Безусловный переход	**BR D** **JUMP D**	CEXX	IP+D+1 → IP

13. Подпрограммы в БЭВМ. Цикл исполнения команд перехода и возврата из подпрограммы. Стек, передача параметров. Позиционно-независимый код. Загрузчик и библиотеки.

Достаточно часто встречаются ситуации, когда отдельные части программы должны выполнить одни и те же действия по обработке данных. В подобных случаях повторяющиеся части программы выделяют в подпрограмму. В базовой ЭВМ для этой цели используются команды CALL и RET.

<структура подпрограммы>



Варианты передачи данных в подпрограмму:

- Аккумулятор (Регистры Общего Назначения)
- Адресуемые ячейки памяти
- Стек
- Регистровые окна

Цикл исполнения CALL

● DR → BR; Адрес перехода записать в BR

● IP → DR; подготовить адрес возврата для записи в стек

● BR → IP; Переход на подпрограмму

● ~0 + SP → SP, AR; уменьшить стек на 1

● DR → MEM(AR); записать адрес возврата

● GOTO INT; Завершение цикла

Цикл исполнения RET

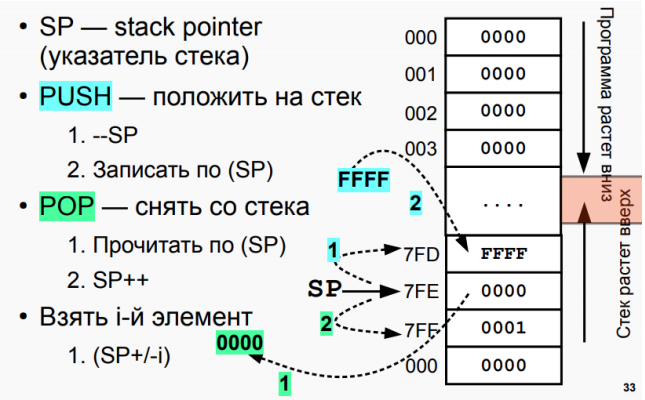
● SP → AR; Вершину стека поместить в AR

● MEM(AR) → DR; прочитать адрес возврата

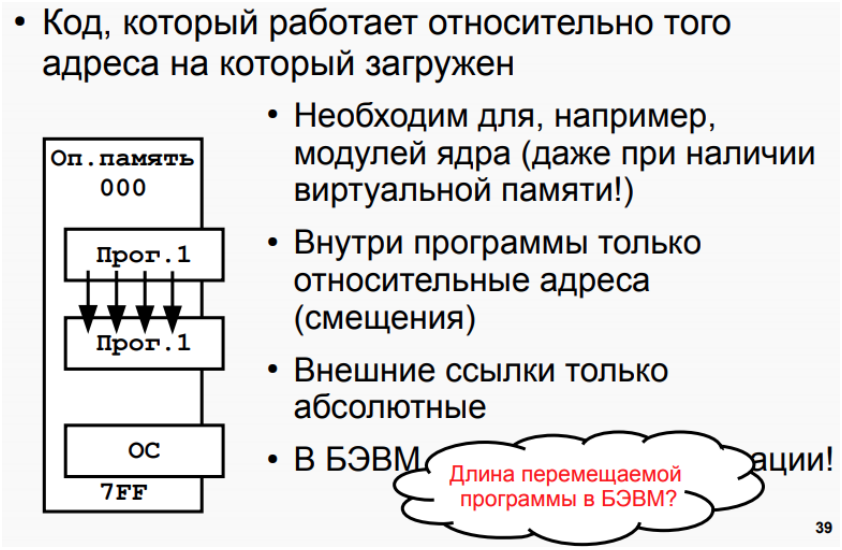
● DR → IP; вернуться из подпрограммы

● SP + 1 → SP; увеличить стек на 1

● GOTO INT; Завершение цикла



PIC - Position Independent Code



Загрузчик и динамический линковщик программ:

Любая ОС имеет соответствующую программу или часть ядра

– Загрузка по выбранному ОС адресу (даже в виртуальной памяти)

– Изменение константных частей адресов в программе

– Загрузка базовых значений регистров

– Динамическая загрузка разделяемых библиотек

– Связывание адресов основной программы с вызываемыми библиотеками

Библиотеки:

● Набор стандартных библиотечных функций

● Разделяемые (динамически линкуемые) и архивные (статически линкуемые)

– # find /lib /usr/lib -name "*.so" |wc -l 3510

– Статические связывают вызовы функций с телом функции в процессе компиляции

– Динамические — в момент загрузки

● Если вам нужна функция — см. в библиотеки

14. Организация ввода-вывода в вычислительных системах. Инициация обмена, передача информации и завершение обмена. Драйверы.

К ЭВМ можно подключать большое число разнообразных устройств ввода-вывода или внешних устройств (ВУ). Эти устройства передают в ЭВМ и получают из нее большой объем информации, который не может быть размещен только в регистрах процессора. Поэтому информация передается из ВУ в память ЭВМ и поступает на ВУ из ее памяти. При этом обмен может идти под управлением программы ЭВМ через регистры процессора (программно-управляемая передача данных) или под управлением специального внешнего устройства (контроллера прямого доступа в память), минуя процессор (передача данных при прямом доступе к памяти).

Программно-управляемый обмен осуществляется малыми порциями, при прямом доступе к памяти информация передается большими блоками. При использовании программно-управляемого обмена должна быть составлена программа, обеспечивающая пересылку данных из памяти ЭВМ в аккумулятор и далее в регистр памяти контроллера ВУ (вывод данных) или из регистра данных контроллера ВУ в аккумулятор и затем в память ЭВМ (ввод данных). В этой программе можно реализовать один из трех типов обмена: синхронный, асинхронный и по прерыванию.

Передача данных: Синхронная/Асинхронная

Завершение обмена и получение драйвером (программой) результата: Синхронное/асинхронное

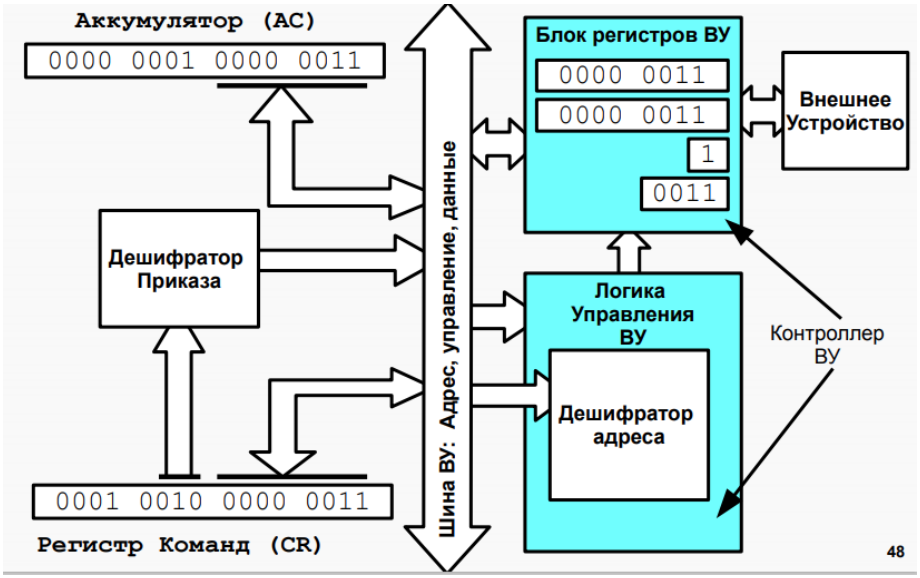
Драйверы:

- Организуют совместную работу с устройством
- «Знают» о принципах работы устройства, адресах регистров, поддерживаемых режимах работы
- Управляются единообразным программным интерфейсом

15. Организация ввода-вывода в БЭВМ. Устройства ввода-вывода, команды.

Команда ввода-вывода

Наименование	Мнемон.	Код	Описание
Запрет прерываний	DI	1000	
Разрешение прерываний	EI	1100	
Ввод	IN REG	12XX	REG → AC
Вывод	OUT REG	13XX	AC → REG
Прерывание	INT NUM	18XX	Програмное прерывание с векторм NUM
Возврат из прерывания	IRET	0B00	(SP)+ → PS, (SP)+ → IP



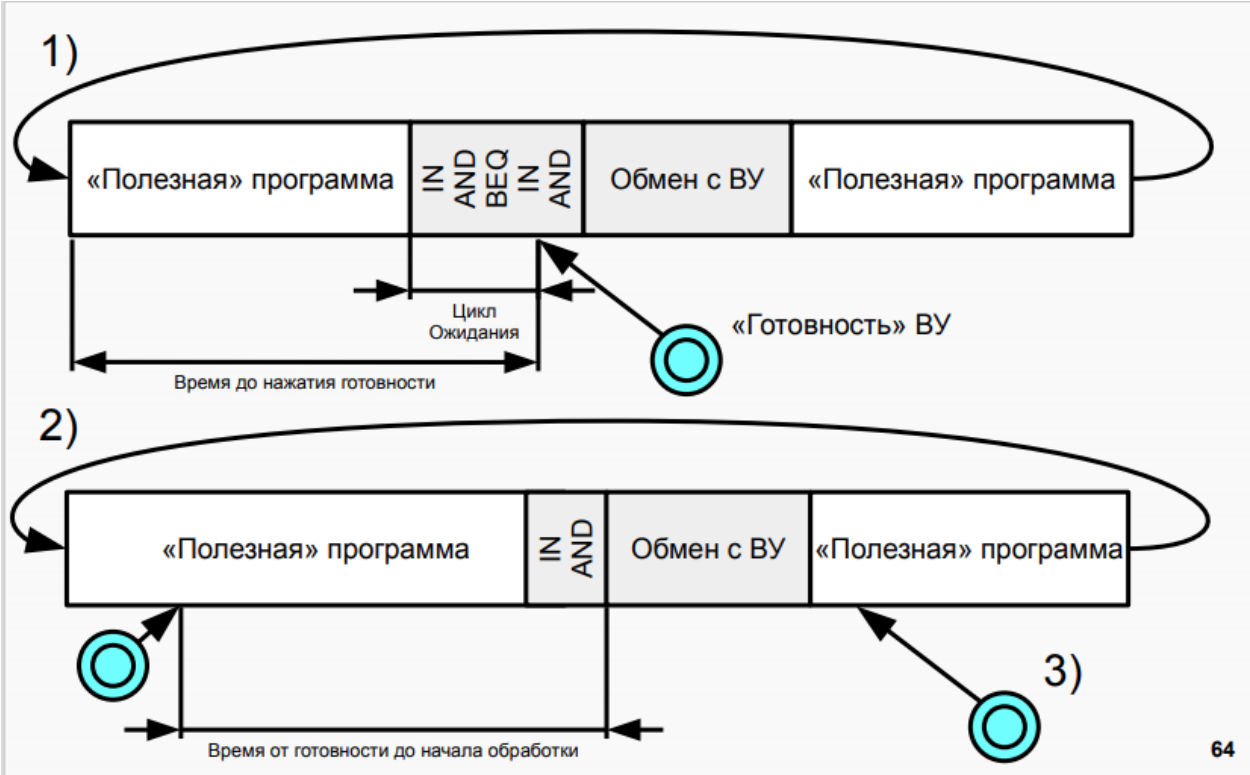
Между ВУ и процессором установлены простейшие контроллеры, каждый из которых содержит: регистр данных (для обмена данными между ВУ и процессором), дешифратор адреса (позволяющий выделить обращение к данному ВУ среди всех обращений у устройствам ввода-вывода, подключенным к процессору), логику управления (декодирующий приказ от процессора на выполнение тех или иных операций) и регистр состояния (в котором хранится информация о готовности ВУ к обмену данными с процессором). В контроллерах простейших ВУ обычно используется однобитовые регистры состояния, которые часто называют флагом. Контроллеры ВУ связаны с процессором шинами ввода и вывода информации, шиной адреса и шиной управления, по которым передаются приказы от процессора и сведения о состоянии ВУ.

Устройства ВУ: вывода, ввода, ввода-вывода, ввода-вывода PRO, таймер, принтер, бегущая строка, 7 разр. индикатор, клавиатура, numpad

16.Организация асинхронного обмена в БЭВМ. Пример программы. Временные издержки асинхронного обмена.

Алгоритм программы асинхронного обмена: сначала проверяется готовность ВУ к обмену и, если оно готово, дается команда на обмен (ввод или вывод). ВУ сообщает о готовности установкой в единицу флага в контроллере ВУ. При асинхронном обмене ЭВМ должна тратить время на ожидание момента готовности, а так как готовность проверяется командным путем, то в это время ЭВМ не может выполнять никакой другой работы по преобразованию данных.

```
        ORG     0x10
START:  CLA
S1:     IN      5       ; Ожидание ввода первого символа
        AND     #0x40   ; Бит 6 SR == 0 («Готов» нажата?)
        BEQ     S1      ; Нет - "Спин-луп"
        IN      4       ; Ввод первого символа
-------------------------------------------------------
        SWAB            ; Сдвиг первого символа
        ST      RES     ; Сохранение его в ячейке RES
```



17.Организация прерываний в БЭВМ. Вектора прерываний, контроллер прерывания.

Прерывание — сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания. В контексте БЭВМ это сигнал о готовности обмена данными с некоторым ВУ.

Также стоит отметить преимущества над асинхронным режимом передачи данных (если Вы помните, то реализовывать его можно двумя способами: через spin-loop или одну проверку с последующим окном ввода-вывода в циклической полезной программе):

- процессор не простаивает в ожидании готовности ВУ
- организованная работа сразу со всеми нужными нам ВУ
- если все ВУ не готовы к обмену, то процессор занят полезной работой

Ярким отличием прерываний является обязательное **сохранение состояния процессора в момент прерывания**, чтобы когда прерывание было обработано, мы смогли вернуться в основную программу и смогли продолжить ее выполнение без разнообразных коллизий.

Адр.	Сод.
0x0	Адр 0
0x1	PS 0
0x2	Адр 1
0x3	PS 1
0x4	Адр 2
0x5	PS 2
0x6	Адр 3
0x7	PS 3
0x8	Адр 4
0x9	PS 4
0xA	Адр 5
0xB	PS 5
0xC	Адр 6
0xD	PS 6
0xE	Адр 7
0xF	PS 7

Вектор прерывания:

● Совокупность адреса программы обработки прерывания и регистра состояния (PS)

● Необходимо инициализировать перед началом обработки прерывания

– Хотя бы установить на подпрограмму, которая ничего не делает

– Ответственность OS и БИОС

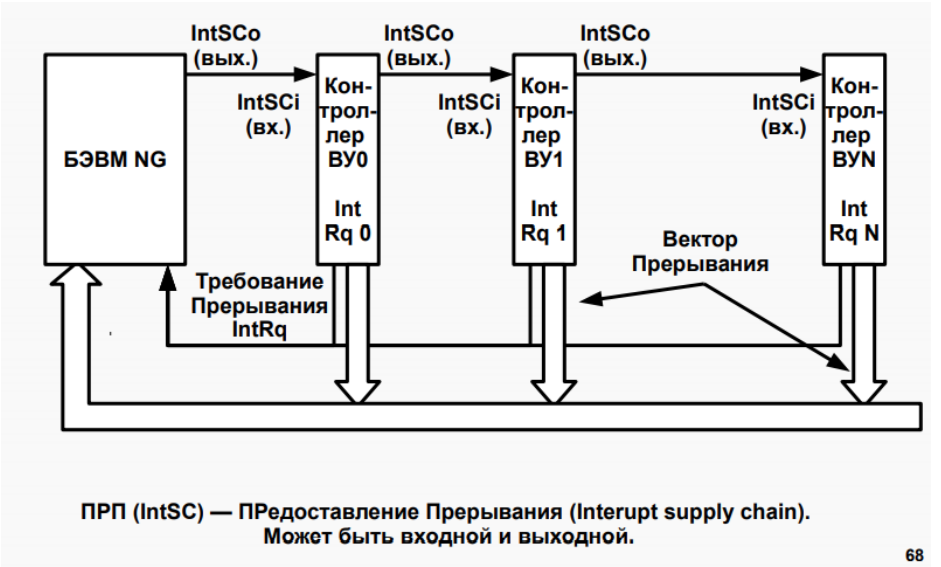
● В БЭВМ-NG ячейки с 0x000 по 0x10

– Всего 8 векторов, по два слова на вектор

– На одном векторе может быть несколько прерываний

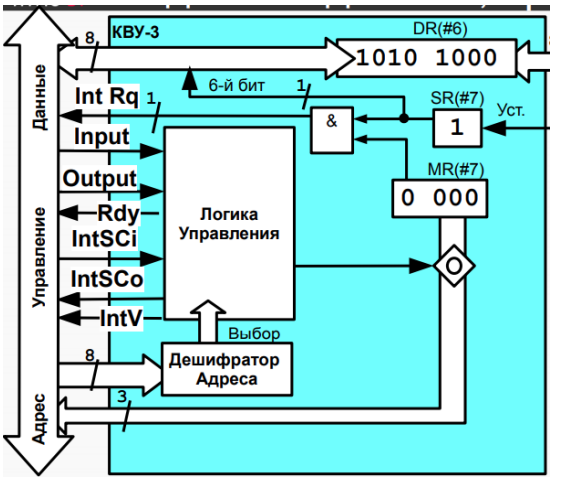
Регистр управления или Management Register — регистр, хранящий в себе информации о разрешении/запрете прерывания от данного ВУ и вектора прерывания, к которому привязано ВУ.

В 3х младших битах содержится номер вектора прерывания, в 3 бите будет 1, есди прерывания от данного ВУ разрешены и 0 — если запрещены.



ПРП (IntSC) — ПРедоставление Прерывания (Interupt supply chain). Может быть входной и выходной.

68



- Сигнал последовательно проходит через контроллеры 0, 1, 2 и останавливается в логике управления КВУ-3
- Внутри ЛУ происходит магия и открывается вентиль у MR (переключаемся на 2 картинку)
- MR подключается к CR через шину адреса и младшие три бита MR записываются в младшие 3 бита CR, при этом оставшиеся 5 бит младшего байта CR заполняются нулями.
- Параллельно с этим ЛУ маскирует сигнал IntSCo и сигнал предоставления прерывания не передается в следующие контроллеры
- Также на шину управления поступает сигнал из ЛУ IntV (Interrupt Vector), который информирует о предоставлении данным контроллером номера вектора прерывания.

18.Организация обмена по прерыванию программы в БЭВМ. Пример программы. Цикл прерывания.

В БЭВМ доступно 8 векторов прерываний, и расположены они в ячейках памяти 0x0 — 0xF включительно. На один вектор прерывания может приходится несколько прерываний. До начала необходимо загрузить вектора прерываний в ВУ и разрешить прерывания.

Цикл прерывания:

1) IF PS(W) == 0: GOTO STOP; Если тумблер РАБОТА\ОСТАНОВ установлен на ОСТАНОВ, то работа БЭВМ прекращается

2) IF PS(IRQ) == 0: GOTO INFETCH; Если запроса на прерывания нет (0 в 6 бите PS), то переходим к циклу выборки СЛЕДУЮЩЕЙ команды

3) IRQSC; Формируется сигнал предоставления прерывания

4) SP + ~0 -> SP, AR;

5) IP -> DR; ///IP -> -(SP)

6) DR -> MEM(AR);

7) SP + ~0 -> SP, AR;

8) PS -> DR; ////PS -> -(SP)

9) DR -> MEM(AR); LTOL(CR) -> BR; Выделили номер вектора прерывания

10) SHL(BR) -> BR, AR; Выделили адрес адреса обработчика прерывания (тавтология, привет)

11) MEM(AR) -> DR; в DR адрес обработчика прерывания

12) DR -> IP

13) LTOL(BR + 1) -> AR ; в AR адрес, в котором лежит PS для обработчика прерывания

14) MEM(AR) -> DR

15) DR -> PS

```
   Готовность ВУ1: 2*A→РДВУ1, Готовность ВУ3: РДВУ3→ яч.3F
         ORG  0x0       ; Инициализация векторов прерывания
V0:      WORD $DEFAULT,0x180 ; Вектор прерывания #0
V1:      WORD $INT1,0x180    ; Вектор прерывания #1
V2:      WORD $DEFAULT,0x180 ; Вектор прерывания #2
V3:      WORD $INT3,0x180    ; Вектор прерывания #3
         …
V7:      WORD $DEFAULT,0x180 ; Вектор прерывания #7
DEFAULT:IRET                 ; Просто возврат
         ORG 0x020 ; Заргузка начальных векторов прерывания
START:   DI
         CLA
         OUT 1     ; MR КВУ-0 на вектор 0
         OUT 5     ; MR КВУ-2 на вектор 0
         LD  #9    ; разрешить прерывания и вектор №1
         OUT 3     ; (1000|0001=1001) в MR КВУ-1
         LD  #0xB  ; разрешить прерывания и вектор №3
         OUT 7     ; (1000|0011=1011) в MR КВУ-3
         ...
         BR  $PROG
```

```
Готовность ВУ1: 2*A→РДВУ1, Готовность ВУ3: РДВУ3→ яч.3F
PROG:
        EI          ; Установка состояния разр. прерывания
        CLA         ; Первоначальная очистка аккумулятора
INCLP:  INC         ; Цикл для наращивания
        BR INCLP    ; содержимого аккумулятора
        ORG 0x03F
        ; Ячейка для хранения кодов, поступающих с ВУ-3
IO3:    WORD  ?
        ORG   0x040
INT1:               ; Прерывание сохранило содержимое PS
        NOP         ; отладочная точка останова (NOP/HLT)
        PUSH        ; Сохранили AC
        ASL         ; Умножили AC на 2
        OUT   2     ; Записали в РДВУ-1 (DR#2)
        POP         ; Вернули AC назад
        NOP         ; отладочная точка останова (NOP/HLT)
        IRET        ; Возврат из обработки прерывания
```

```
Готовность ВУ1: 2*A→РДВУ1, Готовность ВУ3: РДВУ3→ яч.3F
        ORG   0x040
INT3:               ; Прерывание сохранило содержимое PS
        NOP         ; отладочная точка останова (NOP/HLT)
        PUSH        ; AC кладем в стек
        CLA
        IN    6     ; РДВУ-3
        ST    $IO3  ; сохранение
        NOP         ; отладочная точка останова (NOP/HLT)
        POP         ; AC вынимаем из стека
        IRET        ; Возврат из обработки прерывания
```

19. Понятие многоуровневой ЭВМ. Понятие и пример программы на разных уровнях.

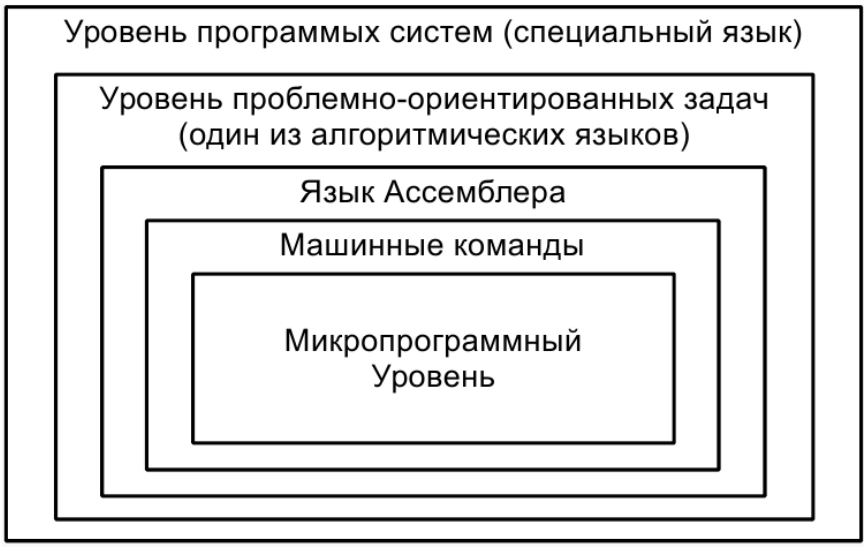
Возможность исполнения на ЭВМ программы, написанной на алгоритмическом языке, обеспечивается с помощью специальных системных программ: компиляторов и интерпретаторов. Компиляция, заключается в том, что процесс выполнения алгоритма осуществляется лишь после завершения процесса перевода исходной программы. В интерпретации же каждый оператор исходной программы заменяется программой-интерпретатором на эквивалентную последовательность машинных команд непосредственно перед исполнением. В отличие от компиляции, в интерпретации во время решения задачи машине нужны и исходная программа, и программа-интерпретатор.

Затраты на создание компиляторов (интерпретаторов) и время на процесс перевода программы в значительной мере определяются сходством компилируемого и получаемого языков. Поэтому алгоритмические языки не сразу переводят программу на язык машинных команд. Существует определенная иерархия языков программирования, в которой более сложный язык базируется на предшествующем.

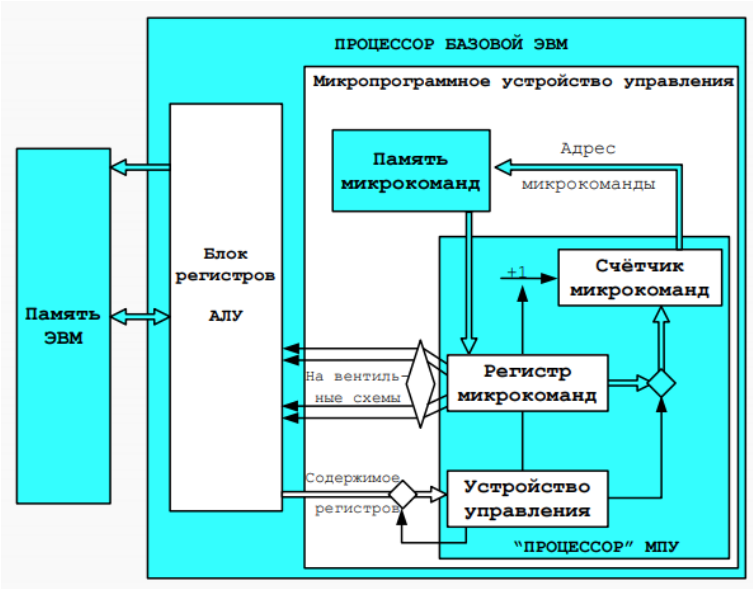
Примером промежуточного языка служит язык символического кодирования команд, часто называемый языком ассемблера. Языки ассемблеров (разработанные для каждого типа ЭВМ) - это первые средства автоматизации программирования в вычислительной технике. В них допускается использование символических имен и меток. Компиляторы с таких языков называются ассемблерами. Они отводят определенные ячейки памяти для символических переменных, организуют связи между различными частями программы, что резко облегчает программирование по сравнению с программированием на уровне команд.

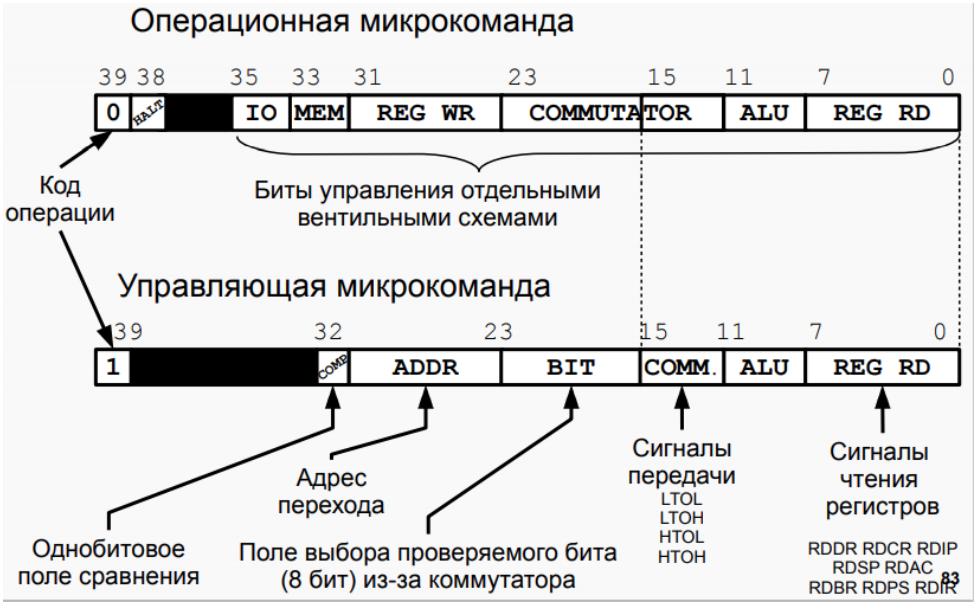
Человеку, работающему с ЭВМ на том или другом языке, чаще всего кажется, что язык, на котором он общается с ЭВМ, – это ее машинный язык. Следовательно, разным пользователям одной и той же ЭВМ может казаться, что они работают на разных вычислительных машинах. Отсюда появились понятия: виртуальная (кажущаяся) ЭВМ и многоуровневая ЭВМ.

Многоуровневая ЭВМ – это вычислительная машина, имеющая средства для работы с n различными уровнями языков программирования. Нижний язык, или уровень, является наиболее простым, верхний –наиболее сложным. Такую машину можно рассматривать как n различных виртуальных машин, каждая из которых имеет свой машинный язык. Сложность аппаратурной реализации этих виртуальных машин возрастает по мере увеличения номера уровня.

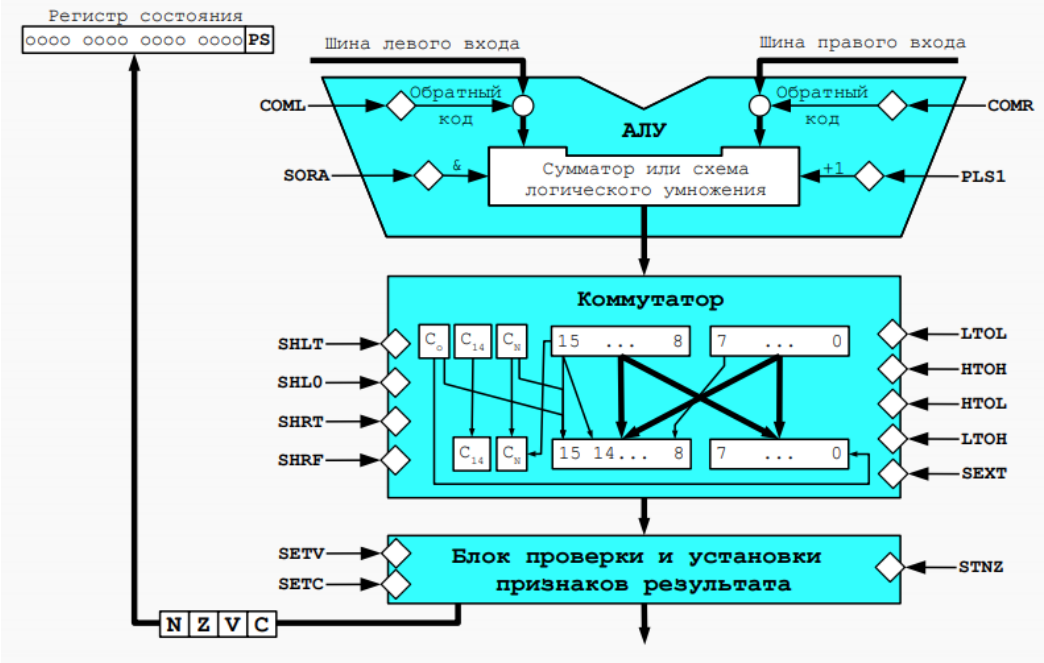


20.Микропрограммный уровень БЭВМ. Структура МПУ. Форматы микрокоманд.





21.Структура и принципы работы арифметико-логического устройства и коммутатора. Регистр состояния БЭВМ



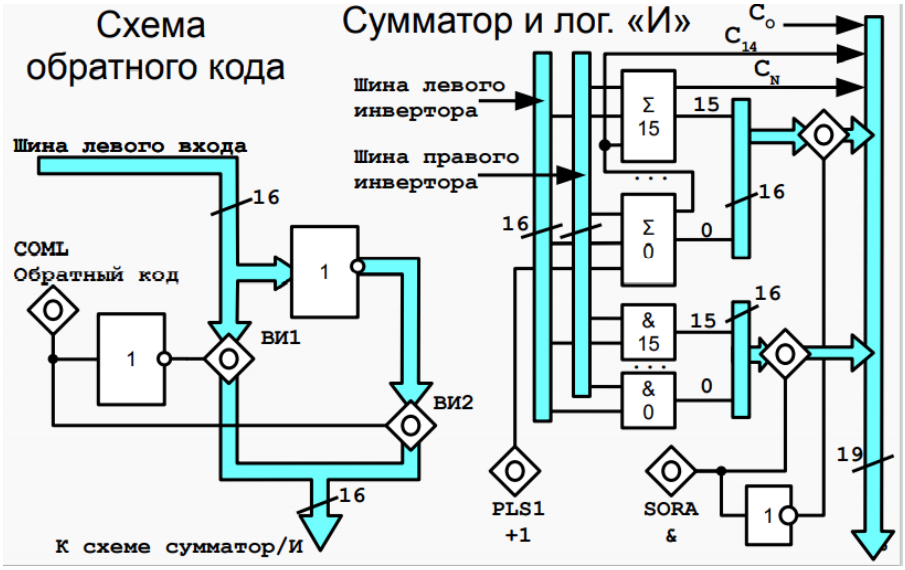
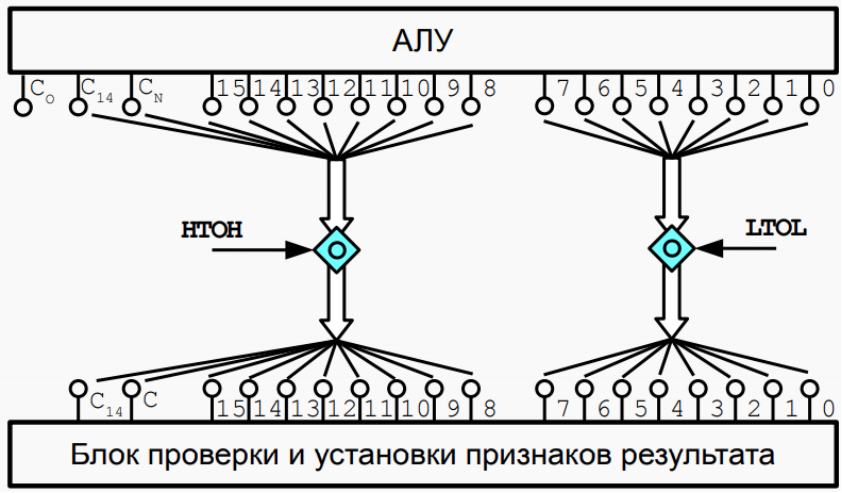
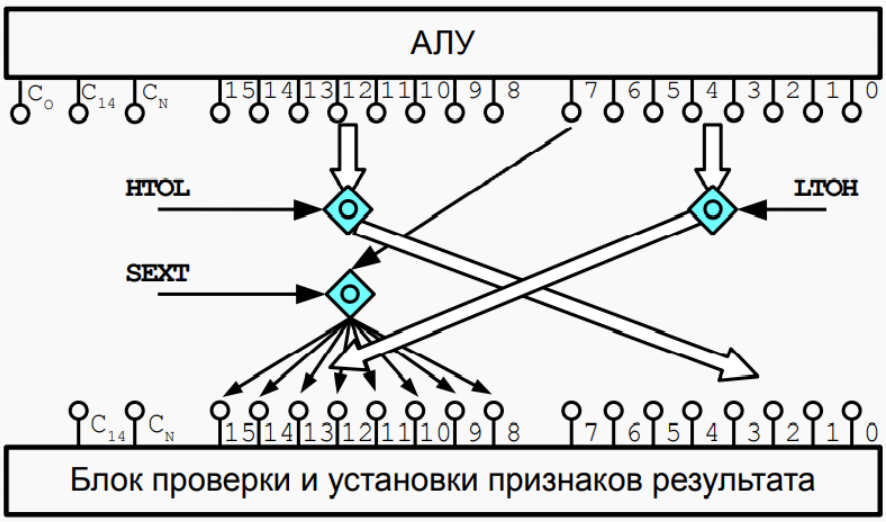
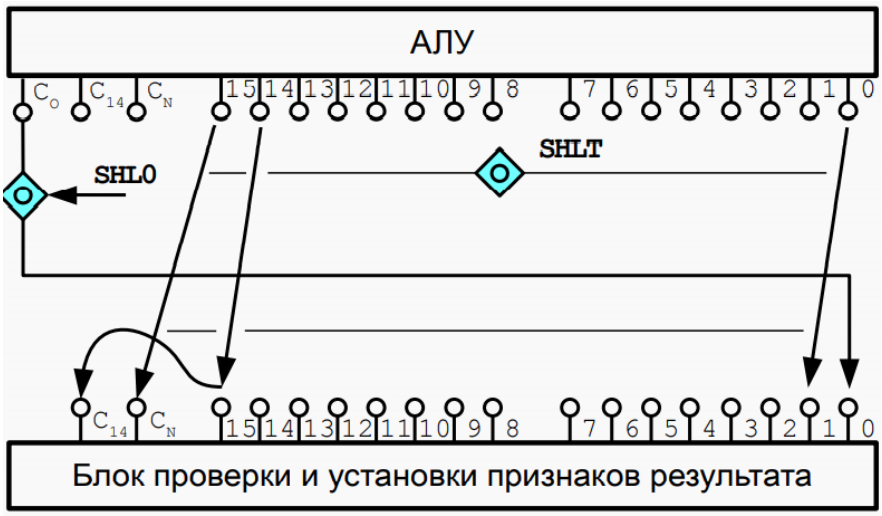
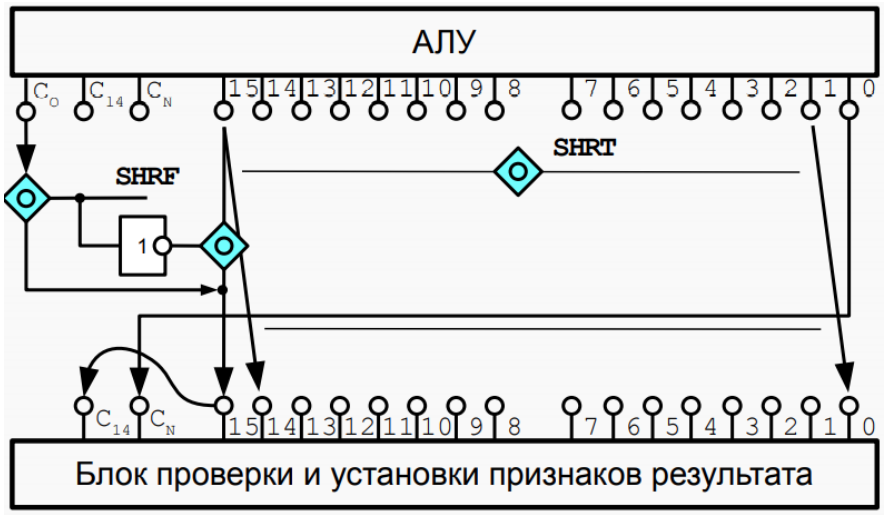


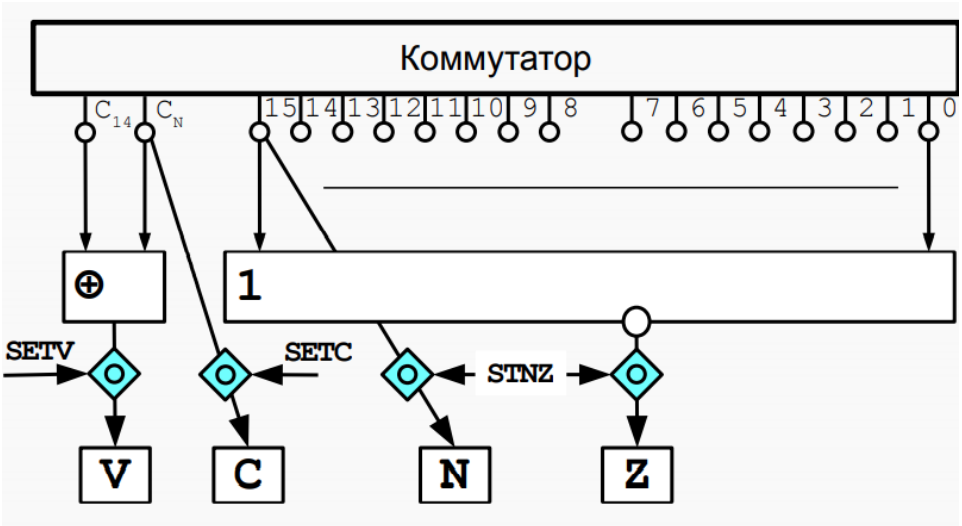
Схема
обратного кода
Сумматор и лог. «И»
Шина левого входа
16
COML
Обратный код
1
1
ВИ1
ВИ2
16
К схеме сумматор/И
Шина левого
инвертора
Шина правого
инвертора
16
Σ
15
Σ
0
15
0
16
&
15
&
0
15
0
16
PLS1
+1
SORA
&
1
19
C_o
C_{14}
C_N

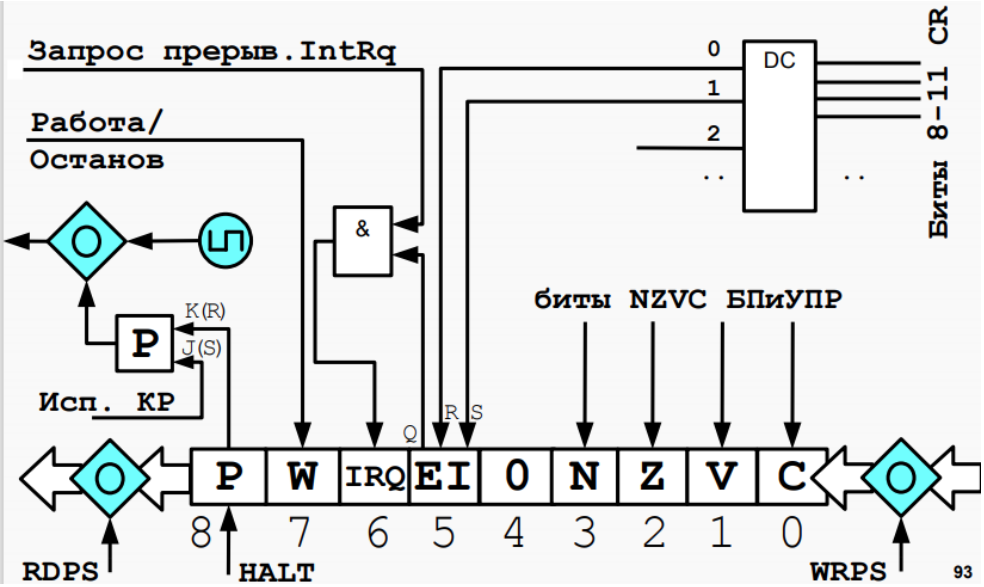

АЛУ
C_o C_{14} C_N 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
HTOH
LTOL
C_{14} C 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Блок проверки и установки признаков результата


АЛУ
C_o C_{14} C_N 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
HTOL
LTOH
SEXT
C_{14} C_N 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Блок проверки и установки признаков результата

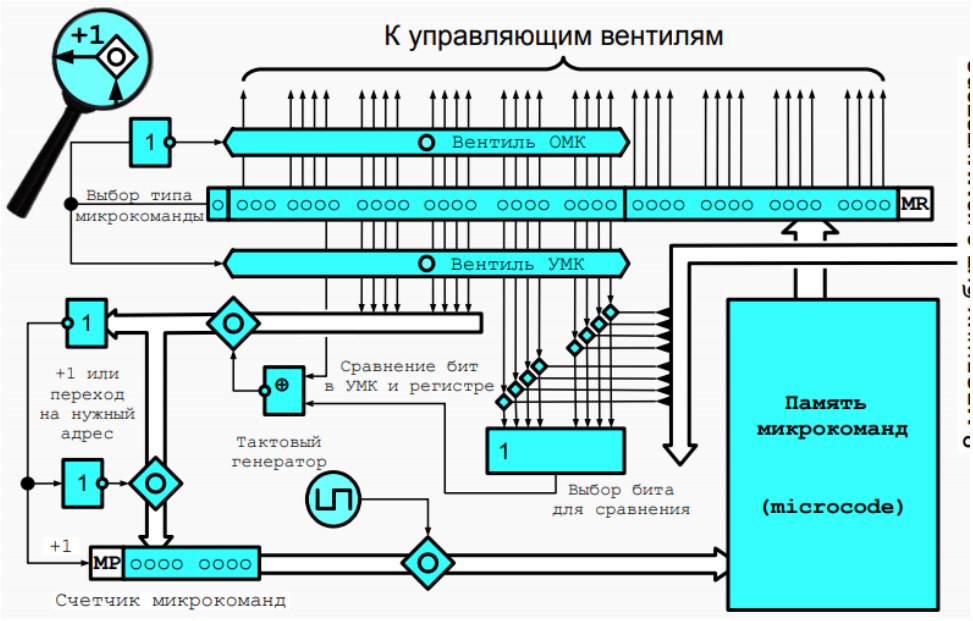

АЛУ
C_O C_{14} C_N 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SHL0
SHLT
C_{14} C_N 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Блок проверки и установки признаков результата


АЛУ
C_O C_{14} C_N 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SHRF
SHRT
1
C_{14} C_N 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Блок проверки и установки признаков результата


Коммутатор
C_{14} C_N 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
⊕
1
SETV
SETC
STNZ
V
C
N
Z



22.Микропрограммное управление вентильными схемами. Схема управления. Интерпретатор БЭВМ.



Интерпретатор БЭВМ

- 256 ячеек для хранения микрокоманд, включая резерв
- Содержит горизонтальные микрокоманды
- Цикл выборки команд
- Цикл выборки адреса операнда и обработки режимов адресации
- Цикл выборки операнда
- Цикл исполнения

– Декодирование и исполнение адресных команд

– Декодирование и исполнение ветвлений

– Декодирование и исполнение безадресных команд

● Декодирование и исполнение команд ввода-вывода

● Цикл прерывания

● Пультовые операции

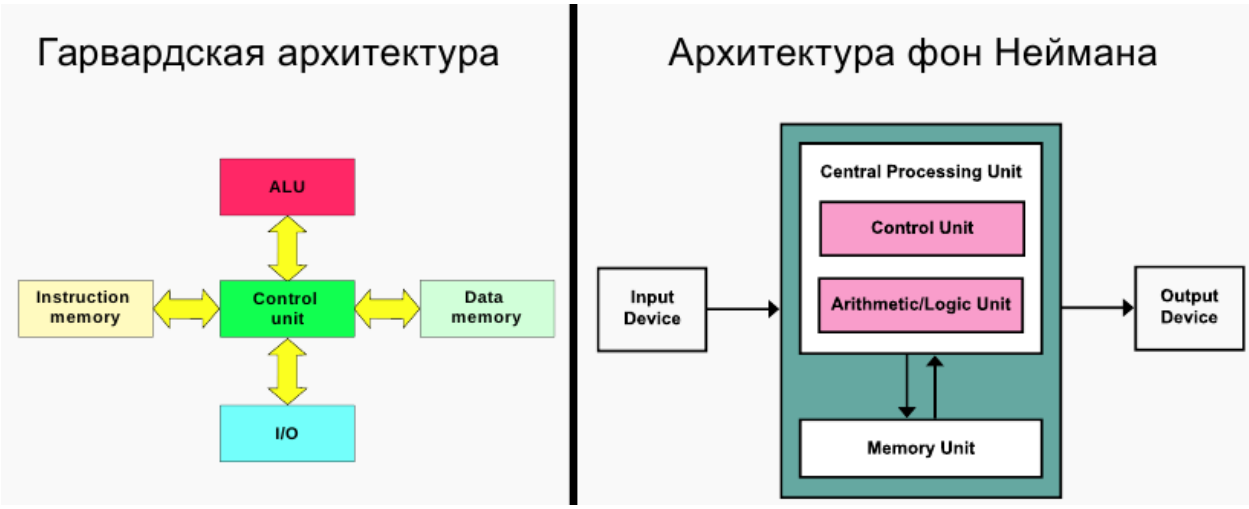
● Свободные ячейки для:

– Арифметической команды

– Команды перехода

– Безадресной команды

23.Архитектура ЭВМ. Гарвардская и фон-Неймановская архитектура. Организация обмена архитектуры ЭВМ с использованием шин.



● Первое поколение — электронные лампы

– Лебедев, 1950, МЭСМ – БЭСМ, 1953, БЭСМ — 10000 оп/с, 53КВТ.

● Второе поколение — транзисторы

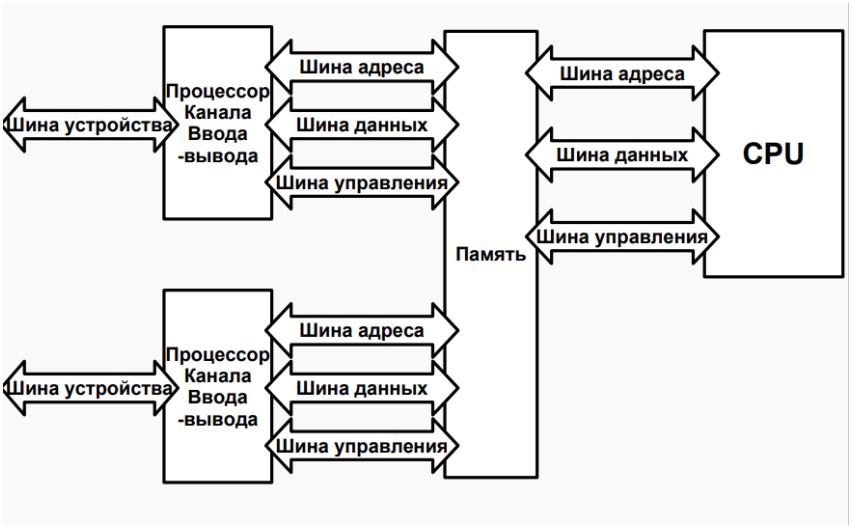
– 5Э92б, 1964, самодиагностика, горячая замена, 500000 оп/с – БЭСМ-6, 1965 год, +ковейерная обработка, удаленное управление по телеф. Линиями

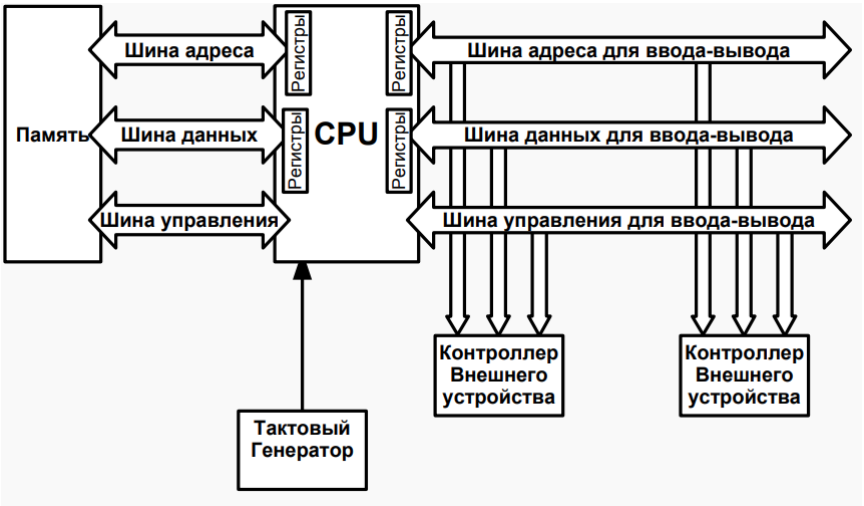
● Третье поколение — интегральные схемы

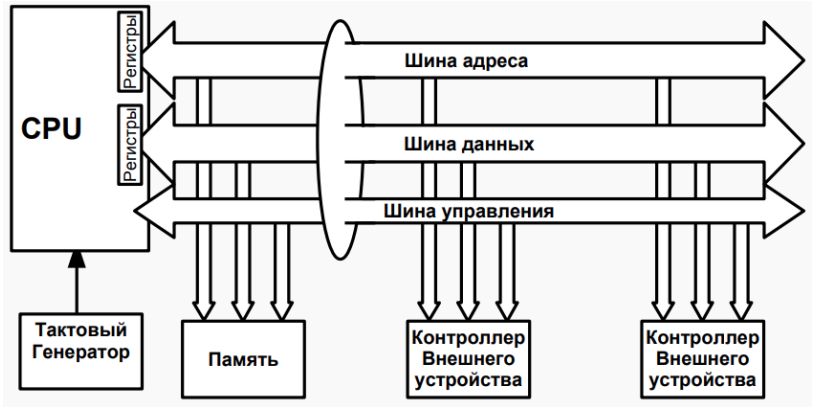
– Директива «Ряд», 1968 год, клонирование S/360, 1971 год — ЕС ЭВМ – Клоны PDP-11

● Четвертое поколение — сверхбольшие интегральные схемы

– Эльбрус — разработка по настоящее время

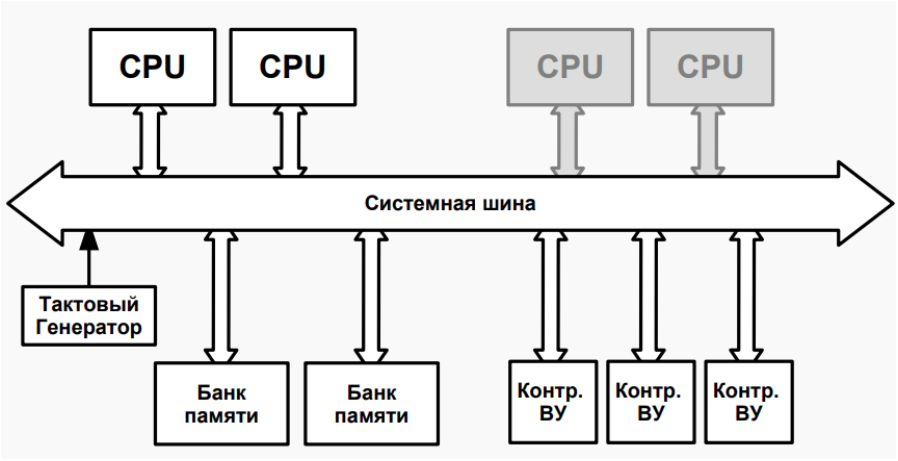


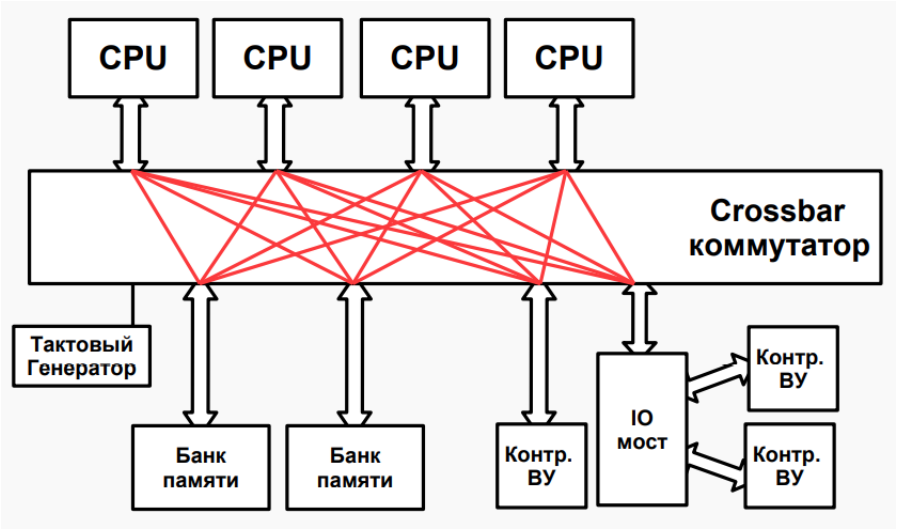




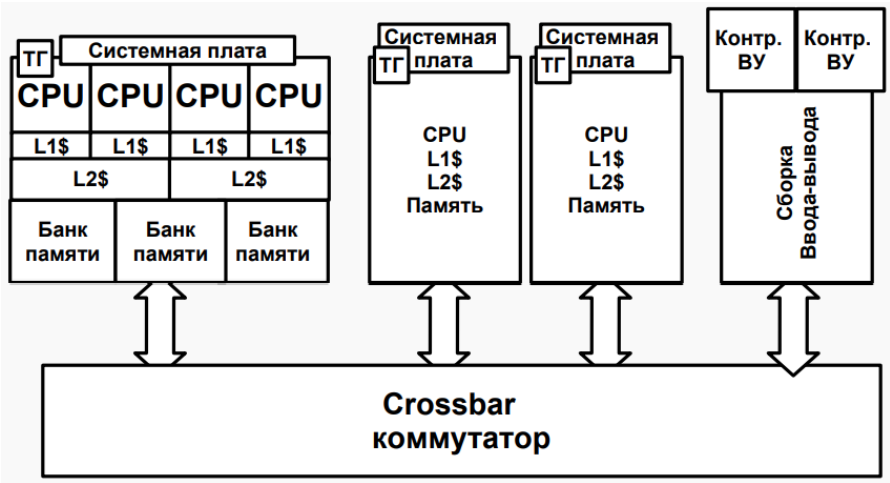
24. Архитектура многопроцессорных ЭВМ. Системный коммутатор. Архитектуры UMA и NUMA.

UMA:





NUMA:



Система **UMA (Uniform Memory Access)** - это архитектура с общей памятью для многопроцессорных систем. В этой модели используется единственная память, к которой обращаются все процессоры представленной многопроцессорной системы с помощью межсоединительной сети. Каждый процессор имеет равное время доступа к памяти (задержка) и скорость доступа. Он может использовать либо одну шину, несколько шин или коммутатор.

NUMA (неоднородный доступ к памяти) также является многопроцессорной моделью, в которой каждый процессор связан с выделенной памятью. Однако эти небольшие части памяти объединяются в единое адресное пространство. Главное, над чем подумать, это то, что в отличие от UMA время доступа к памяти зависит от расстояния, на котором расположен процессор, что означает изменение времени доступа к памяти. Это позволяет получить доступ к любой ячейке памяти, используя физический адрес.

Ключевые различия между UMA и NUMA

1. Модель UMA (совместно используемая память) использует один или два контроллера памяти. В отличие от этого, NUMA может иметь несколько контроллеров памяти для доступа к памяти.
2. В архитектуре UMA используются одиночные, множественные и перекрестные шины. И наоборот, NUMA использует иерархические и древовидные типы шин и сетевых подключений.
3. В UMA время доступа к памяти для каждого процессора одинаково, в то время как в NUMA время доступа к памяти изменяется по мере изменения расстояния памяти от процессора.
4. Приложения общего назначения и разделения времени подходят для машин UMA. В отличие от этого, подходящее приложение для NUMA ориентировано в режиме реального времени и критично ко времени.
5. Параллельные системы на основе UMA работают медленнее, чем системы NUMA.
6. Когда речь идет о пропускной способности UMA, имеют ограниченную пропускную способность. Напротив, NUMA имеет пропускную способность больше, чем UMA.

25. Структура современных процессоров. Окружение процессора. CISC, RISC, VLIW.

● Разрядность адреса и данных 16/32/64 бита

● Тактовые частоты 500МГЦ-5Ггц.

● Многопроцессорные 1-100+ CPU

● Многоядерные 1-16 ядер

● От 1 ГБ до терабайтов ОЗУ

● Используют кэш-память разных уровней

● Суперскалярные

● CISC, RISC, VLIW

● Complex Instruction Set Computer

– Традиционные процессоры (например Intel), отягощенные совместимостью

- наличие в процессоре сравнительно небольшого числа регистров общего назначения;

- большое количество машинных команд, некоторые из них аппаратно реализуют сложные операторы ЯВУ;
- разнообразие способов адресации операндов;
- множество форматов команд различной разрядности; наличие команд, где обработка совмещается С обращением к памяти.

● Reduced Instruction Set Computer

– Простой набор инструкций, выполнение инструкции за такт

Идея заключается в ограничении списка команд ВМ наиболее часто используемыми простейшими командами, оперирующими данными, размещенными только в регистрах процессорах. Обращение к памяти допускается лишь с помощью специальных команд чтения и записи. Резко уменьшено количество форматов команд и способов указания адресов операндов. Направлено на быстродействие.

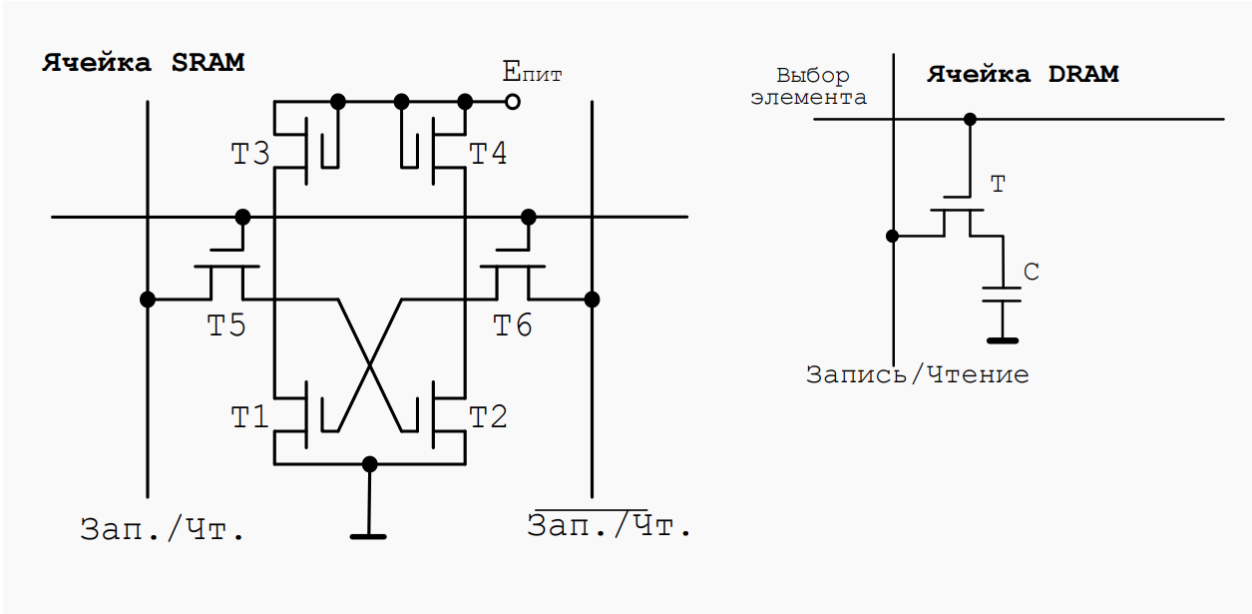
● Very Long Instructions Word

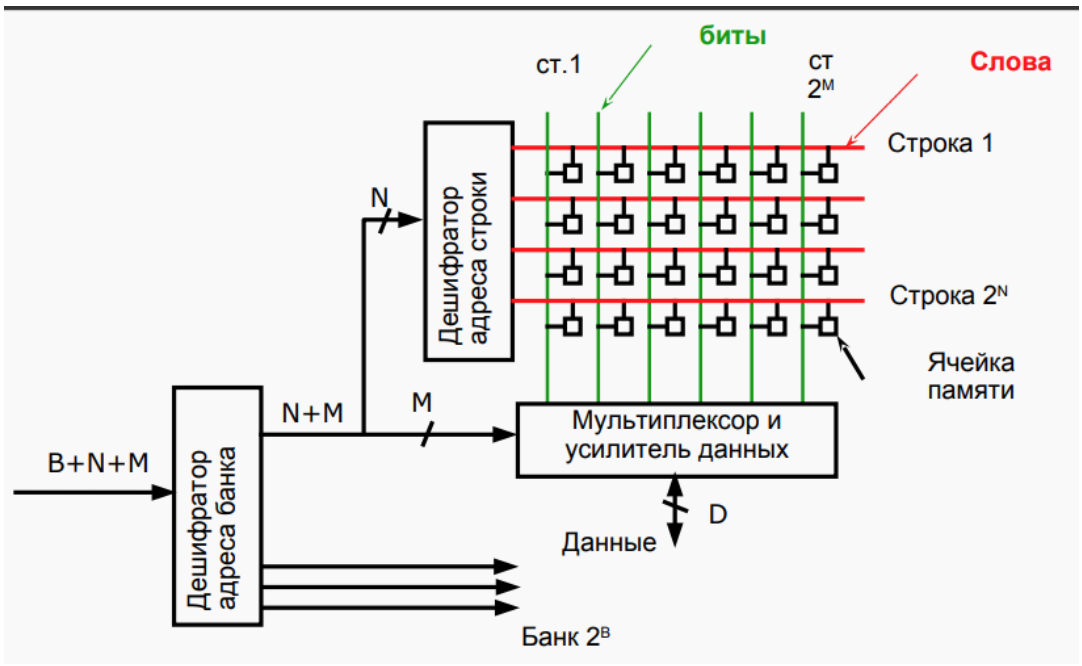
– Несколько инструкций, упакованных в одну команду

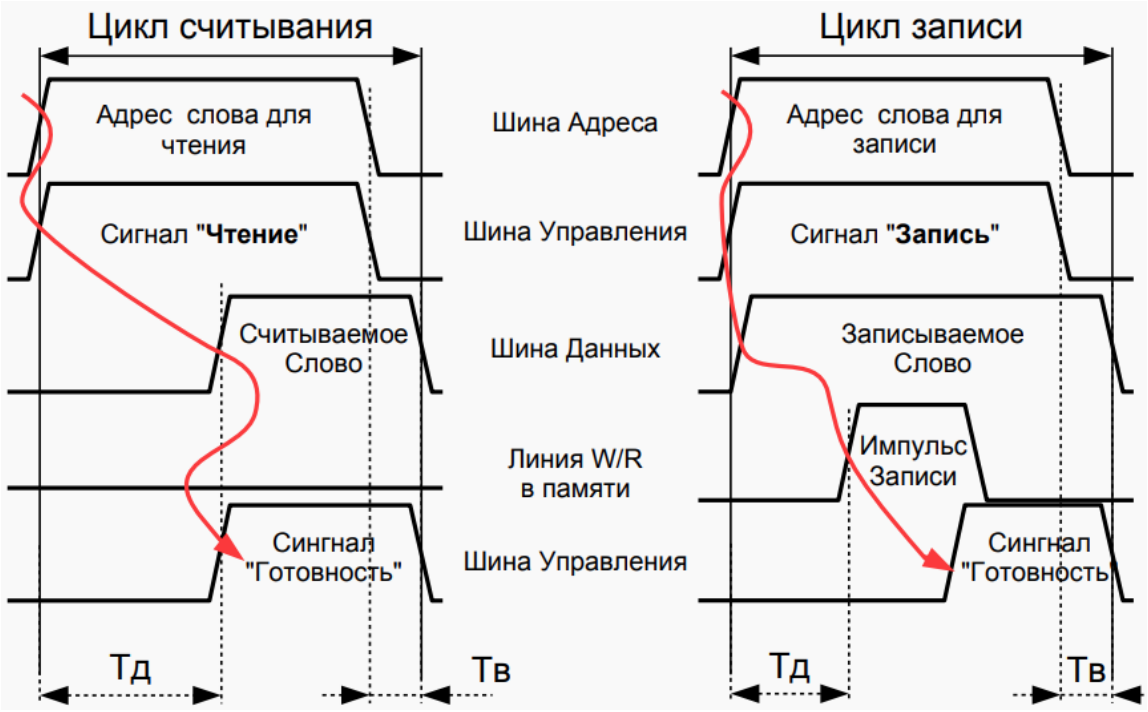
– Упаковка операций в инструкцию ложится на компилятор

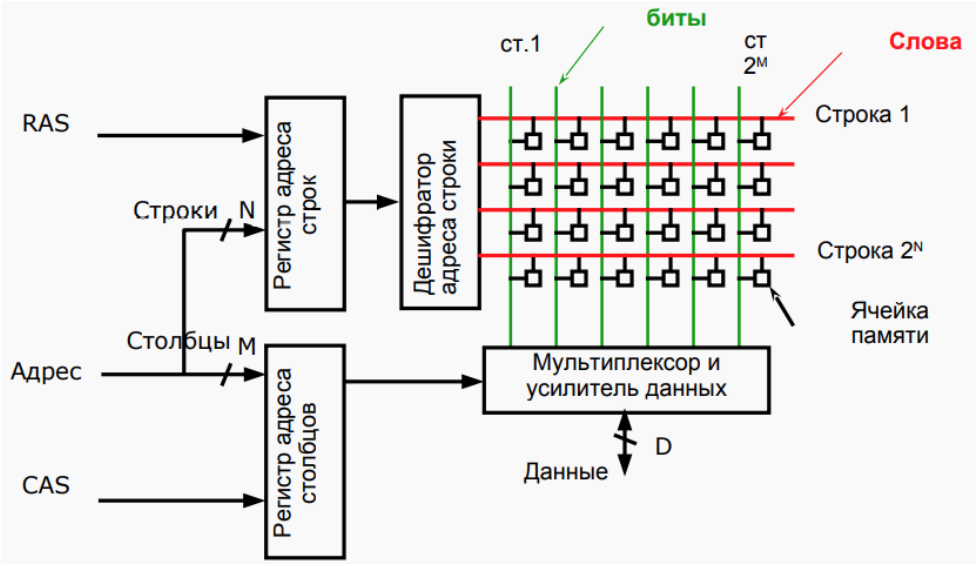
Концепция VLIW базируется на RISC-архитектуре, где несколько простых RISC-команд объединяются в одну сверхдлинную команду и выполняются параллельно.
Идея VLIW базируется на том, что задача эффективного планирования параллельного выполнения нескольких команд возлагается на «разумный» компилятор. Такой компилятор вначале исследует исходную программу с целью обнаружить все команды, которые могут быть выполнены одновременно, причем так, чтобы это не приводило к возникновению конфликтов. В процессе анализа компилятор может даже частично имитировать выполнение рассматриваемой программы. На следующем этапе компилятор пытается объединить такие команды в пакеты, каждый из которых рассматривается так одна сверхдлинная команда.

26.Адресуемая память, организация и временные диаграммы. Конструктивные особенности современной памяти.




биты
ст.1
ст
2M
Слова
Строка 1
Строка 2N
Ячейка
памяти
Дешифратор
адреса строки
N
N+M
M
B+N+M
Дешифратор
адреса банка
Мультиплексор и
усилитель данных
D
Данные
Банк 2B


Цикл считывания
Цикл записи
Адрес слова для
чтения
Шина Адреса
Адрес слова для
записи
Сигнал "Чтение"
Шина Управления
Сигнал "Запись"
Считываемое
Слово
Шина Данных
Записываемое
Слово
Импульс
Записи
Линия W/R
в памяти
Сингнал
"Готовность"
Шина Управления
Сингнал
"Готовность"
Тд
Тв
Тд
Тв



Конструктивные особенности современной памяти:

● Burst mode — пакетный режим

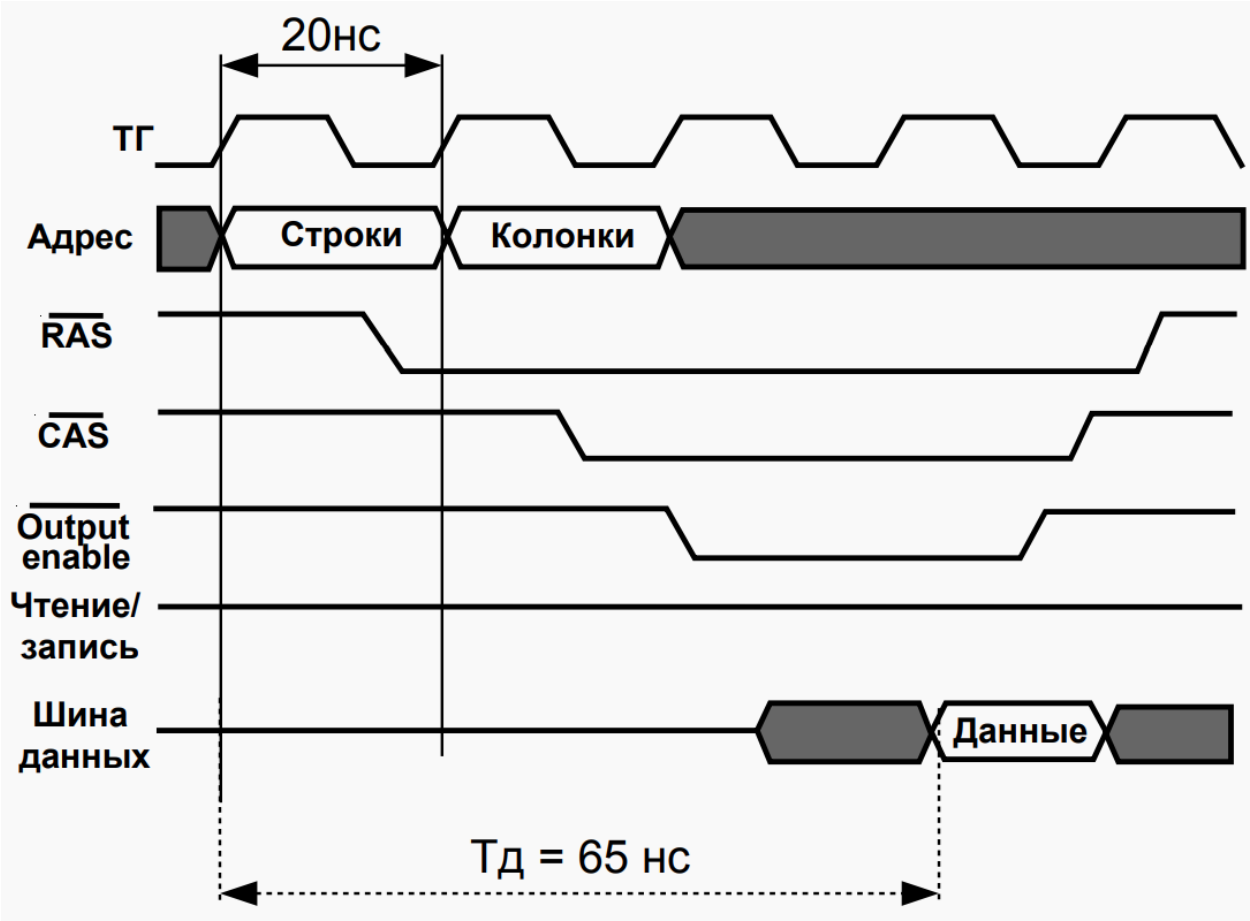
● Double Data Rate — передача данных и по фронту и по спаду

● SPD — чип, содержащий идентификационную информацию

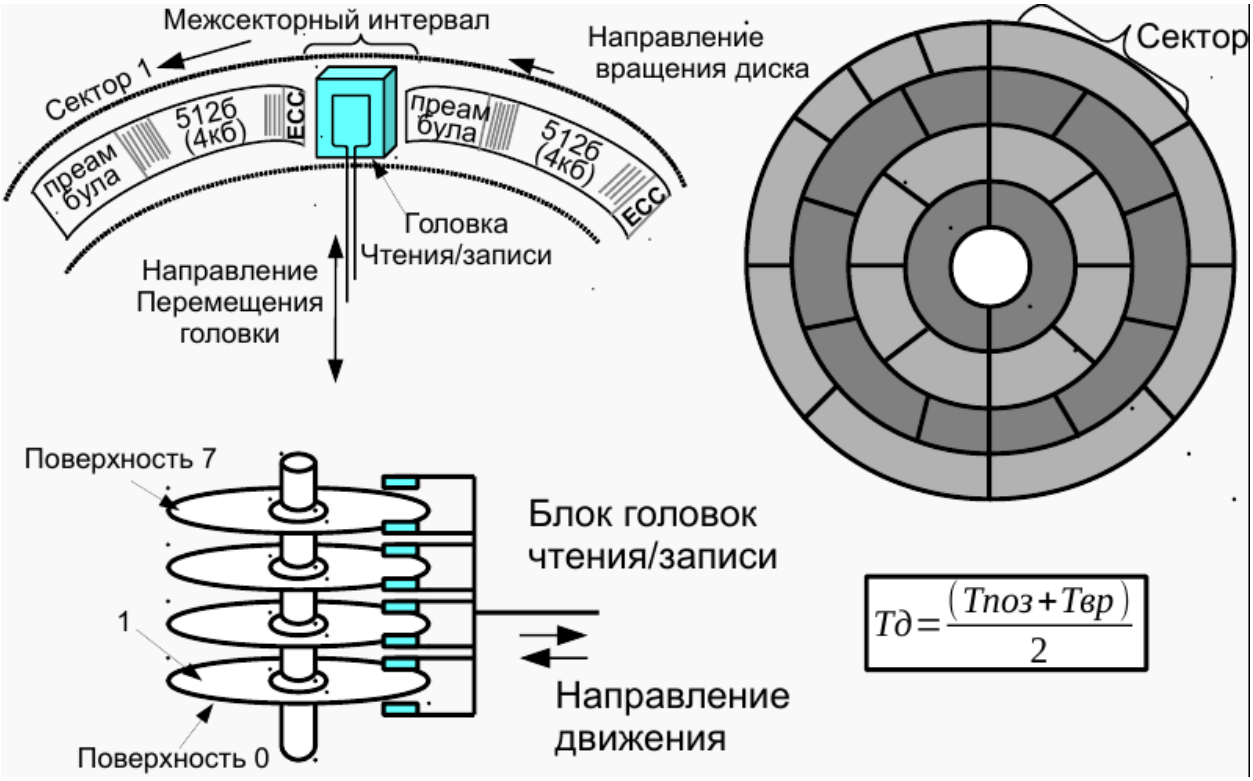
● Interleaving — расслоение памяти, повышает производительность

● DDR4-2133 8192MB **PC4-17000** индекс производительности

- включение в микросхемы динамической памяти некоторого количества статической памяти;
- ***синхронная работа*** памяти и ЦП, т.е. использование внутренней ***конвейерной архитектуры*** и ***чередование адресов***.
- ***пакетный режим*** (Burst Mode) – в нем проц. запрашивает данные из памяти не отдельными байтами, а в виде пакетов, состоящих из 32 или 64 бит, т.е. считывает 2 или 4 машинных слова за 1 такт;
- ***чередование памяти*** (Interleaving Mode) – основано на том, что логические связанные байты располагаются в памяти друг за другом. В момент регенерации памяти она становится недоступной для процессора. Чтобы не было таких пауз в работе памяти, данные помещаются в различные микросхемы памяти. Таким образом, пока в одной из микросхем происходит регенерация данных, процессор в это время может считывать байты из другой микросхемы;
- ***разбиение памяти на страницы*** (Paging Mode) – основан на том же принципе, что и предыдущий режим. Все ячейки с одинаковым адресом строки группируются в т.н. *страницы.* Поскольку адреса строк не изменяются, то это экономит время на поиск и считывание данных. Обычно память делится на страницы размером 512 и более байт;
- ***кэширование памяти*** – используется для ускорения доступа к данным, находящимся в RAM. Для этой цели между CPU и RAM ставится небольшая кэш-память, обычно в пределах 2 Мбайт, которая работает на частоте процессора, а значит при обращении к ней не требуются циклы ожидания.



27.Память, ориентированная на записи (блочная память). Организация дисковой памяти и памяти на магнитных лентах.



28.Характеристики запоминающих устройств. Пирамида памяти.

- Месторасположение – процессорные, внутренние, внешние
- Емкость – В метрических (Кило-) и двоичных (Киби-) множителях
- Единица пересылки – Слово, строка кэша, блок на диске
- Метод доступа – Произвольный (адресный), ориентированных на записи (прямой), последовательный, ассоциативный

Характеристики памяти

1. Быстродействие и временные соотношения
    a. Время доступа Тд
    b. Длительность цикла памяти (время обращения) Тц
    c. Время чтения и время записи
    d. Время восстановления Тв
    e. Скорость передачи информации
2. Физический тип и особенности
3. Стоимость

Запоминающие устройства имеют ряд показателей качества, характеризующих их информационные и временные свойства.

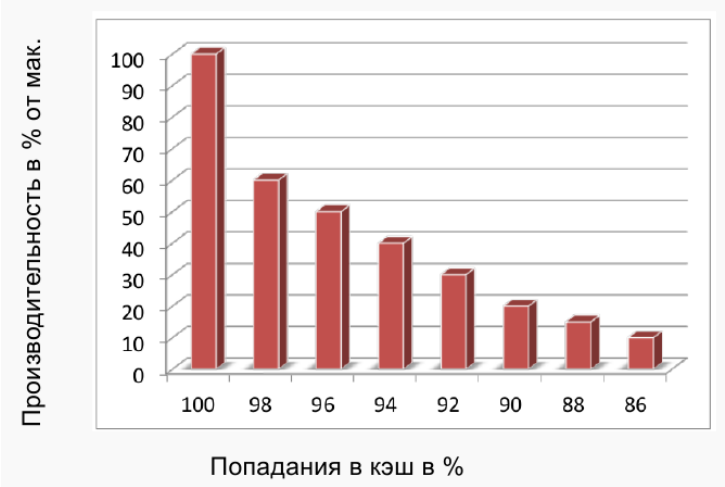
Информационная емкость памяти выражается в количестве битов, байтов или слов, состоящих из определенного числа байтов.

Время доступа – временной интервал, определяемый от момента, когда процессор выставил на адресной шине адрес требуемой ячейки и послал по шине управления приказ на чтение или запись данных, до момента осуществления связи адресуемой ячейки с шиной данных.

Время записи – интервал времени, необходимый для переписи содержимого шины данных в связанную с ней ячейку памяти.

	Объем	Тд	*	Тип	Управл.
CPU	100-1000 б.	<1нс	1с	Регистр	компилятор
L1 Cache	32-128Кб	1-4нс	2с	Ассоц.	аппаратура
L2-L3 Cache	0.5-32Мб	8-20нс	19с	Ассоц.	аппаратура
Основная память	0.5Гб-4ТБ	60-200нс	50-300с	Адресная	программно
SSD	128Гб-1Тб/drive	25-250мкс	5д	Блочн.	программно
Жесткие диски	0.5Тб-4Тб/drive	5-20мс	4м	Блочн.	программно
Магнитные ленты	1-6Тб/к	1-240с	200л	Последов.	программно

29. Ассоциативная память, Кэш-память. Влияние промахов кэш-памяти на производительность.



Процессоры всегда работали быстрее, чем память. Процессоры и память совершенствовались параллельно, поэтому это несоответствие сохранялось. Поскольку на микросхему можно помещать все больше и больше транзисторов, разработчики процессоров использовали эти преимущества для создания конвейеров и супер-скалярной архитектуры, что еще больше повышало скорость работы процессоров. Разработчики памяти обычно использовали новые технологии для увеличения емкости, а не скорости, что еще больше усугубляло проблему. На практике такое несоответствие в скорости работы приводит к следующему: после того как процессор дает запрос памяти, должно пройти много циклов, прежде чем он получит слово, которое ему нужно. Чем медленнее работает память, тем дольше процессору приходится ждать, тем больше циклов должно пройти.

Есть два пути решения этой проблемы. Самый простой из них — начать считывать информацию из памяти, когда это необходимо, и при этом продолжать выполнение команд, но если какая-либо команда попытается использовать слово до того, как оно считалось из памяти, процессор должен приостанавливать работу. Чем медленнее работает память, тем чаще будет возникать такая проблема и тем больше будет проигрыш в работе. Например, если отсрочка составляет 10 циклов,

весьма вероятно, что одна из 10 следующих команд попытается использовать слово, которое еще не считалось из памяти.

Другое решение проблемы — сконструировать машину, которая не приостанавливает работу, но следит, чтобы программы-компиляторы не использовали слова до того, как они считаются из памяти. Однако это не так просто осуществить на практике. Часто при выполнении команды загрузки машина не может выполнять другие действия, поэтому компилятор вынужден вставлять пустые команды, которые не производят никаких операций, но при этом занимают место в памяти. В действительности при таком подходе простаивает не аппаратное, а программное обеспечение, но снижение производительности при этом такое же.

На самом деле эта проблема не технологическая, а экономическая. Инженеры знают, как построить память, которая будет работать так же быстро, как и процессор, но при этом ее приходится помещать прямо на микросхему процессора (поскольку информация через шину поступает очень медленно). Установка большой памяти на микросхему процессора делает его больше и, следовательно, дороже, и даже если бы стоимость не имела значения, все равно существуют ограничения в размерах процессора, который можно сконструировать. Таким образом, приходится выбирать между быстрой памятью небольшого размера и медленной памятью большого размера. Мы бы предпочли память большого размера с высокой скоростью работы по низкой цене. Маленькая память с высокой скоростью работы называется **кэш-памятью** (от французского слова cacher «прятать»). Основная идея кэш-памяти проста: в ней находятся слова, которые чаще всего используются. Если процессору нужно какое-нибудь слово, сначала он обращается к кэш-памяти. Только в том случае, если слова там нет, он обращается к основной памяти. Если значительная часть слов находится в кэш-памяти, среднее время доступа значительно сокращается. Таким образом, успех или неудача зависит от того, какая часть слов находится в кэш-памяти. Давно известно, что программы не обращаются к памяти наугад. Если программе нужен доступ к адресу A, то скорее всего после этого ей понадобится доступ к адресу, расположенному поблизости от A. Практически все команды обычной программы (за исключением команд перехода и вызова процедур) вызываются из последовательных участков памяти. Кроме того, большую часть времени программа тратит на циклы, когда ограниченный набор команд выполняется снова и снова. Точно так же при манипулировании матрицами программа скорее всего будет обращаться много раз к одной и той же матрице, прежде чем перейдет к чему-либо другому.

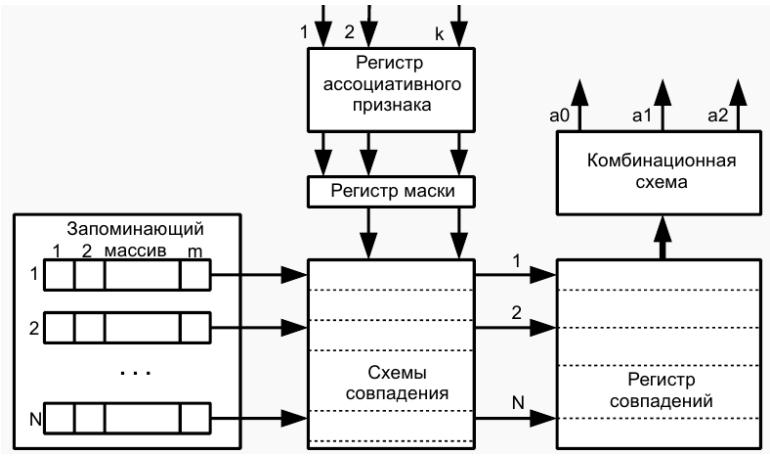
То, что при последовательных отсылках к памяти в течение некоторого промежутка времени используется только небольшой ее участок, называется принципом локальности. Этот принцип составляет основу всех систем кэш-памяти. Идея состоит в следующем: когда определенное слово вызывается из памяти, оно вместе с соседними словами переносится в кэш-память, что позволяет при очередном запросе быстро обращаться к следующим словам. Если слово считывается или записывается k раз, компьютеру понадобится сделать 1 обращение к медленной основной памяти и к-1 обращений к быстрой кэш-памяти. Чем больше к, тем выше общая производительность.

Одной из самых эффективных технологий одновременного увеличения пропускной способности и уменьшения времени ожидания является применение нескольких блоков кэш-памяти. Основная технология — введение отдельной кэш-памяти для команд и отдельной для данных (разделенной кэш-памяти). Такая кэш-память имеет несколько преимуществ. Во-первых, операции могут начинаться независимо в каждой кэш-памяти, что удваивает пропускную способность системы памяти. Именно по этой причине в микроархитектуре Mic-1 нам понадобились два отдельных порта памяти: особый порт для каждой кэш-памяти. Отметим, что каждая кэш-память имеет независимый доступ к основной памяти.

В настоящее время многие системы памяти гораздо сложнее этих. Между разделенной кэш-памятью и основной памятью часто помещается кэш-память второго уровня. Вообще говоря, может быть три и более уровней кэш-памяти, поскольку требуются более продвинутые системы. Прямо на микросхеме центрального процессора находится небольшая кэш-память для команд и небольшая кэш-память для данных, обычно от 16 до 64 Кбайт. Есть еще кэш-память второго уровня, которая расположена не на самой микросхеме процессора, а рядом с ним в том же блоке. Эта кэш-память обычно не является разделенной и содержит смесь данных и команд. Ее размер — от 512 Кбайт до 1 Мбайт. Кэш-память третьего уровня находится на той же плате, что и процессор, и обычно состоит из статического ОЗУ в несколько мегабайтов, которое функционирует гораздо быстрее, чем динамическое ОЗУ основной памяти. Обычно все содержимое кэш-памяти первого уровня находится в кэш-памяти второго уровня, а все содержимое кэш-памяти второго уровня находится в кэш-памяти третьего уровня.

Ассоциативная память (АП) или **ассоциативное запоминающее устройство** (АЗУ) - специальный вид машинной памяти, используемый в приложениях очень быстрого поиска. Известна также под терминами «память, адресуемая по содержимому», «ассоциативное запоминающее устройство», «контентно-адресуемая память» или «ассоциативный массив», хотя последний термин чаще используется в программировании для обозначения структуры данных

В ассоциативной памяти элементы выбираются не по адресу, а по содержимому. Поясним последнее понятие более подробно. Для памяти с адресной организацией было введено понятие минимальной адресуемой единицы (МАЕ) как порции данных, имеющей индивидуальный адрес. Введем аналогичное понятие для ассоциативной памяти, и будем эту минимальную единицу хранения в ассоциативной памяти называть строкой ассоциативной памяти (СтрАП). Каждая СтрАП содержит два поля: поле тега (англ. tag — ярлык, этикетка, признак) и поле данных. Запрос на чтение к ассоциативной памяти словами можно выразить следующим образом: выбрать строку (строки), у которой (у которых) тег равен заданному значению. Особо отметим, что при таком запросе возможен один из трех результатов: имеется в точности одна строка с заданным тегом; имеется несколько строк с заданным тегом; нет ни одной строки с заданным тегом. Поиск записи по признаку — это действие, типичное для обращений к базам данных, и поиск в базе зачастую чвляется ассоциативным поиском. Для выполнения такого поиска следует просмотреть все записи и сравнить заданный тег с тегом каждой записи. Это можно сделать и при использовании для хранения записей обычной адресуемой памяти (и понятно, что это потребует достаточно много времени — пропорционально количеству хранимых записей!). Об ассоциативной памяти говорят тогда, когда ассоциативная выборка данных из памяти поддержана аппаратно.



Принцип работы кэша процессора

Итак, мы разобрались с назначением кэша процессора, а теперь рассмотрим базовые принципы работы кэша, которые позволяют ему решать свою основную задачу.

Кэш состоит из контроллера и собственно кэшпамяти. Кэш-контроллер управляет работой кэшпамяти, то есть загружает в нее нужные данные из оперативной памяти и возвращает, когда нужно, модифицированные процессором данные в оперативную память. Архитектурно кэшконтроллер расположен между процессором и оперативной памятью (рис. 1). Перехватывая запросы к оперативной памяти, кэшконтроллер определяет, имеется ли копия затребованных данных в кэше. Если такая копия там есть, то это называется кэш-попаданием (cache hit) — в таком случае данные очень быстро извлекаются из кэша (существенно быстрее, чем из оперативной памяти). Если же требуемых данных в кэше нет, то говорят о кэш-промахе (cache miss) — тогда запрос данных переадресуется к оперативной памяти.

Рис. 1. Структура кэш-памяти процессора

Для достижения наивысшей производительности кэшпромахи должны происходить как можно реже (в идеале — отсутствовать). Учитывая, что по емкости кэшпамять намного меньше оперативной памяти, добиться этого не такто просто. А потому основная задача кэш-контроллера заключается в том, чтобы загружать кэшпамять действительно нужными данными и своевременно удалять из нее данные, которые больше не понадобятся. Важно понимать, что кэш всегда «полон», так как оставлять часть кэшпамяти пустой нерационально. Новые данные попадают в кэш только путем вытеснения (замещения) какихлибо старых данных.

Загрузка кэша данными реализуется на основе так называемой стратегии кэширования, а выгрузка данных — на основе политики замещения.

30. Предназначение и организация виртуальной памяти. Сегментно-страничная организация. Устройство управления памятью (MMU), буфер трансляции (TLB).

Виртуальная память – технология управления памятью ЭВМ, благодаря которому операционная система может обращаться к памяти, большей, чем память, фактически установленная в компьютере. Это достигается за счет помещения данных в свободное дисковое пространство внешнего ЗУ, которое задействовано в роли оперативной памяти. Необходимо понимать, что часть программ, которые мы не смогли разместить в оперативной памяти из-за её нехватки, теперь будут размещены на ВЗУ и это будет эквивалентно размещению в оперативной памяти. Использование ВЗУ очень удобно, так как в это время пользователь оперирует с общим адресным пространством и ему безразлично, какая физическая память при этом используется: внешняя или внутренняя.

В большинстве современных ОС виртуальная память организуется с помощью страничной адресации. ОП делится на страницы: области памяти фиксированной длины, которые являются минимальной единицей выделяемой памяти. Процесс обращается к памяти с помощью адреса виртуальной памяти, который содержит в себе номер страницы и смещение внутри страницы. Если страница выгружена из ОП, то ОС подкачивает страницу с жёсткого диска. При запросе на выделение памяти операционная система может «сбросить» на жёсткий диск страницы, к которым давно не было обращений. Критические данные (например, код запущенных и работающих программ, код и память ядра системы) обычно находятся в оперативной памяти.

Сегментная организация – механизм организации виртуальной памяти, при котором виртуальное пространство делится на части произвольного размера — сегменты. Для каждого сегмента, как и для страницы, могут быть назначены права доступа к нему пользователя и его процессов. При загрузке процесса часть сегментов помещается в оперативную память, а часть сегментов размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки. Недостатком данного метода распределения памяти является фрагментация на уровне сегментов и более медленное по сравнению со страничной организацией преобразование адреса.

Чтобы добавить поддержку виртуальной памяти, достаточно между процессором и оперативной памятью разместить MMU, которое будет транслировать виртуальные адреса (адреса, используемые в программе) в физические (адреса, попадающие на вход микросхем памяти). Такое расположение очень удобно — MMU используется только тогда, когда процессор обращается к памяти (например, при промахе кэша), а все остальное время не используется и экономит электроэнергию. Кроме того, в этом случае MMU почти не влияет на быстродействие процессора.

Выглядит этот процесс так:

1. Процессор подает на вход MMU виртуальный адрес
2. Если MMU выключено или если виртуальный адрес попал в нетранслируемую область, то физический адрес просто приравнивается к виртуальному
3. Если MMU включено и виртуальный адрес попал в транслируемую область, производится трансляция адреса, то есть замена номера виртуальной страницы на номер соответствующей ей физической страницы (смещение внутри страницы одинаковое):
    - Если запись с нужным номером виртуальной страницы есть в TLB, то номер физической страницы берется из нее же
    - Если нужной записи в TLB нет, то приходится искать ее в таблицах страниц, которые операционная система размещает в нетранслируемой области ОЗУ (чтобы не было промаха TLB при обработке предыдущего промаха). Поиск может быть реализован как аппаратно, так и программно — через обработчик исключения, называемого

страничной ошибкой (page fault). Найденная запись добавляется в TLB, после чего команда, вызвавшая промах TLB, выполняется снова.

TLB:

● Кэширует часто используемые преобразования

● Обычно раздельный для адреса и данных

● Организован в виде ассоциативной памяти

Если запись с нужным номером виртуальной страницы есть в TLB [Translation Lookaside Buffer], то номер физической страницы берётся из нее же

Если нужной записи в TLB нет, то приходится искать ее в таблицах страниц, которые операционная система размещает в нетранслируемой области ОЗУ (чтобы не было промаха TLB при обработке предыдущего промаха). Поиск может быть реализован как аппаратно, так и программно — через обработчик исключения, называемого страничной ошибкой (page fault). Найденная запись добавляется в TLB, после чего команда, вызвавшая промах TLB, выполняется снова.

31. Сетевые технологии, Понятие сети ЭВМ, классификация компьютерных сетей. Сообщение и пакет. Модель взаимодействия открытых систем.

Компьютерная сеть – это совокупность компьютеров, между которыми возможен информационный обмен без промежуточных носителей информации.

Компьютерная сеть – сложная система аппаратных и программных компонентов, взаимосвязанных друг с другом.

Программные компоненты состоят из сетевых ОС и сетевых приложений (почтовые программы, сетевые БД).

Все устройства, подключаемые к сети (т.е. *аппаратные компоненты*) можно разделить на три функциональные группы:

— рабочие станции;
— серверы сети;
— коммуникационные узлы.

Рабочая станция (workstation) – это ПК, подключенный к сети, на котором пользователь сети выполняет свою работу. Каждая рабочая станция обрабатывает свои локальные файлы и использует свою ОС.

Сервер сети (server) – это компьютер, подключенный к сети и предоставляющий пользователям сети определённые услуги, например, хранение данных общего пользования, обработку запросов к СУБД, печать заданий, удалённую обработку заданий и т.д. Можно выделить следующие

группы серверов: файловый сервер, сервер БД, сервер прикладных программ, факс сервер и др.

К коммуникационным узлам относятся следующие устройства:

— повторители;

— коммутаторы (мосты);

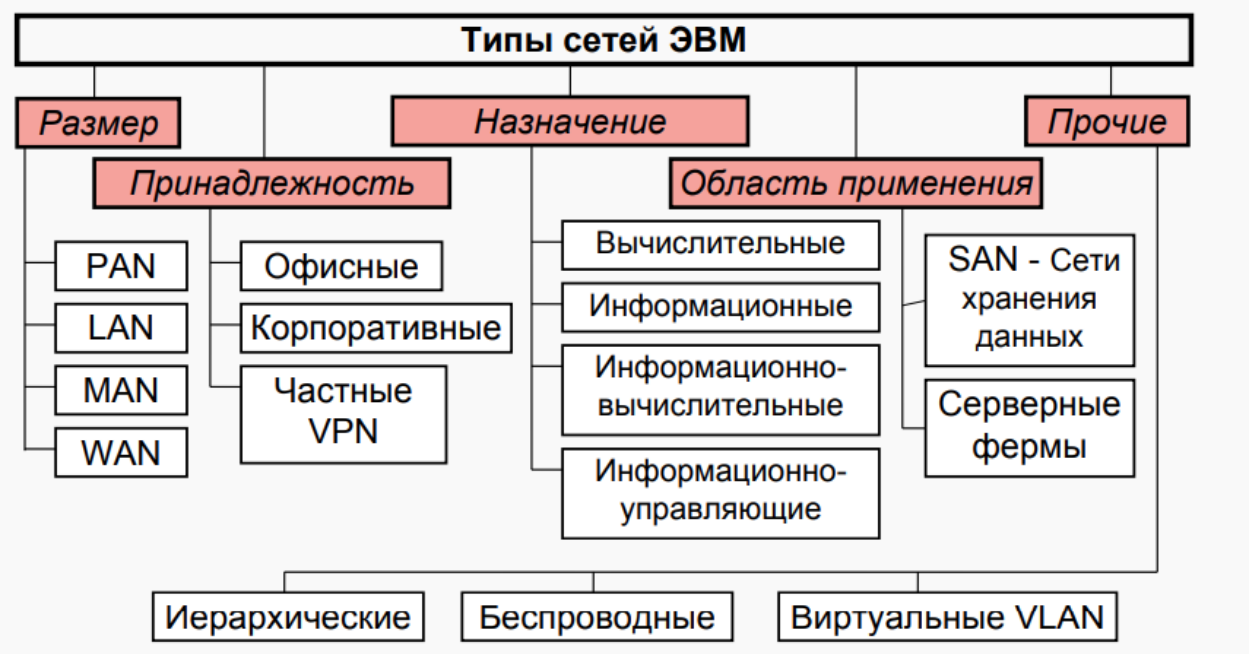
— маршрутизаторы;

— шлюзы.

Это устройства, необходимые для соединения различных сетей друг с другом.

Вычислительная сеть создается для обеспечения потенциального доступа к любому ресурсу сети для любого пользователя сети.

Классификация сетевых технологий:

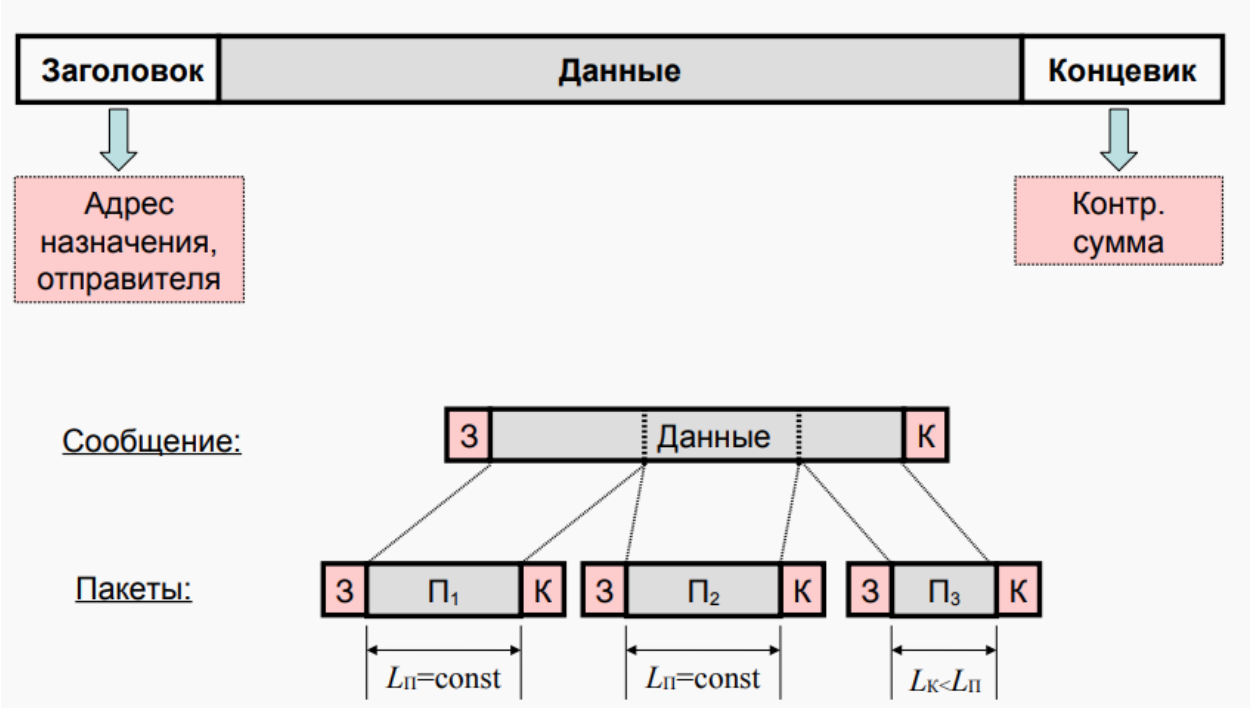


Коммутация сообщений – разбиение информации на сообщения, каждый из которых состоит из заголовка и информации.

Это способ взаимодействия, при котором создается логический канал, путем последовательной передачи сообщений через узлы связи по адресу, указанному в заголовке сообщения.

Коммутация пакетов - это особый способ коммутации узлов сети, который специально создавался для наилучшей передачи компьютерного трафика (пульсирующего трафика).

При коммутации пакетов все передаваемые пользователем сети сообщения разбиваются в исходном узле на сравнительно небольшие части, называемые пакетами. Необходимо уточнить, что сообщением называется логически завершенная порция данных - запрос на передачу файла, ответ на этот запрос, содержащий весь файл, и т. п. Сообщения могут иметь произвольную длину, от нескольких байт до многих мегабайт. Напротив, пакеты обычно тоже могут иметь переменную длину, но в узких пределах, например от 46 до 1500 байт (**EtherNet**). Каждый пакет снабжается заголовком, в котором указывается адресная информация, необходимая для доставки пакета узлу назначения, а также номер пакета, который будет использоваться узлом назначения для сборки сообщения.



Состоит она из 7 уровней и каждый уровень выполняет определенную ему роль и задачи. Разберем, что делает каждый уровень снизу вверх:

1) Физический уровень (Physical Layer): определяет метод передачи данных, какая среда используется (передача электрических сигналов, световых импульсов или радиоэфир), уровень напряжения, метод кодирования двоичных сигналов.

2) Канальный уровень (Data Link Layer): он берет на себя задачу адресации в пределах локальной сети, обнаруживает ошибки, проверяет целостность данных. Если слышали про MAC-адреса и протокол «Ethernet», то они располагаются на этом уровне.

3) Сетевой уровень (Network Layer): этот уровень берет на себя объединения участков сети и выбор оптимального пути (т.е. маршрутизация). Каждое сетевое устройство должно иметь уникальный сетевой адрес в сети. Думаю, многие слышали про протоколы IPv4 и

IPv6. Эти протоколы работают на данном уровне.

4) Транспортный уровень (Transport Layer): Этот уровень берет на себя функцию транспорта. К примеру, когда вы скачиваете файл с Интернета, файл в виде сегментов отправляется на Ваш компьютер. Также здесь вводятся понятия портов, которые нужны для указания назначения к конкретной службе. На этом уровне работают протоколы TCP (с установлением соединения) и UDP (без установления соединения).

5) Сеансовый уровень (Session Layer): Роль этого уровня в установлении, управлении и разрыве соединения между двумя хостами. К примеру, когда открываете страницу на веб-сервере, то Вы не единственный посетитель на нем. И вот для того, чтобы поддерживать сеансы со всеми пользователями, нужен сеансовый уровень.

6) Уровень представления (Presentation Layer): Он структурирует информацию в читабельный вид для прикладного уровня. Например, многие компьютеры используют таблицу кодировки ASCII для вывода текстовой информации или формат jpeg для вывода графического изображения.

7) Прикладной уровень (Application Layer): Наверное, это самый понятный для всех уровень. Как раз на этом уроне работают привычные для нас приложения — e-mail, браузеры по протоколу HTTP, FTP и остальное.

32.Модель TCP/IP: передающая среда, канальный и сетевой уровень. Адресация, передача и маршрутизация пакетов.

Модель TCP/IP:

- Прикладной уровень
- Транспортный уровень
- Сетевой уровень
- Канальный уровень
- Среда передачи: витая пара, коаксиальный кабель, оптика, радиосигналы

Уровень передающей среды:

● Коаксиальный кабель (устарел)

– «толстый» - 10Base-5 — до 500м

– «тонкий» - 10Base-2 — до 50м

● Витая пара 10Base-T, 100Base-T, ….

– Категория 3: от 10 до 100 Мбит/с 100BASE-T4 (100м).

– Категория 5e: 100 Мбит/с (2 пары), 1Гбит/с на (4пары)

– Категория 6: 10 Гбит/с (55м)

– Категория 7a: 40Гбит/с (50м), 100Гбит/с (15м)

● Оптика (10BASE-F,100BASE-SX,10GBASE-ER...)

– ST (Straight Tip)

– SC (Standard Connector)

– LC (Lucent Connector)

– Лазер находится в SFP (Small Plugin Factor)

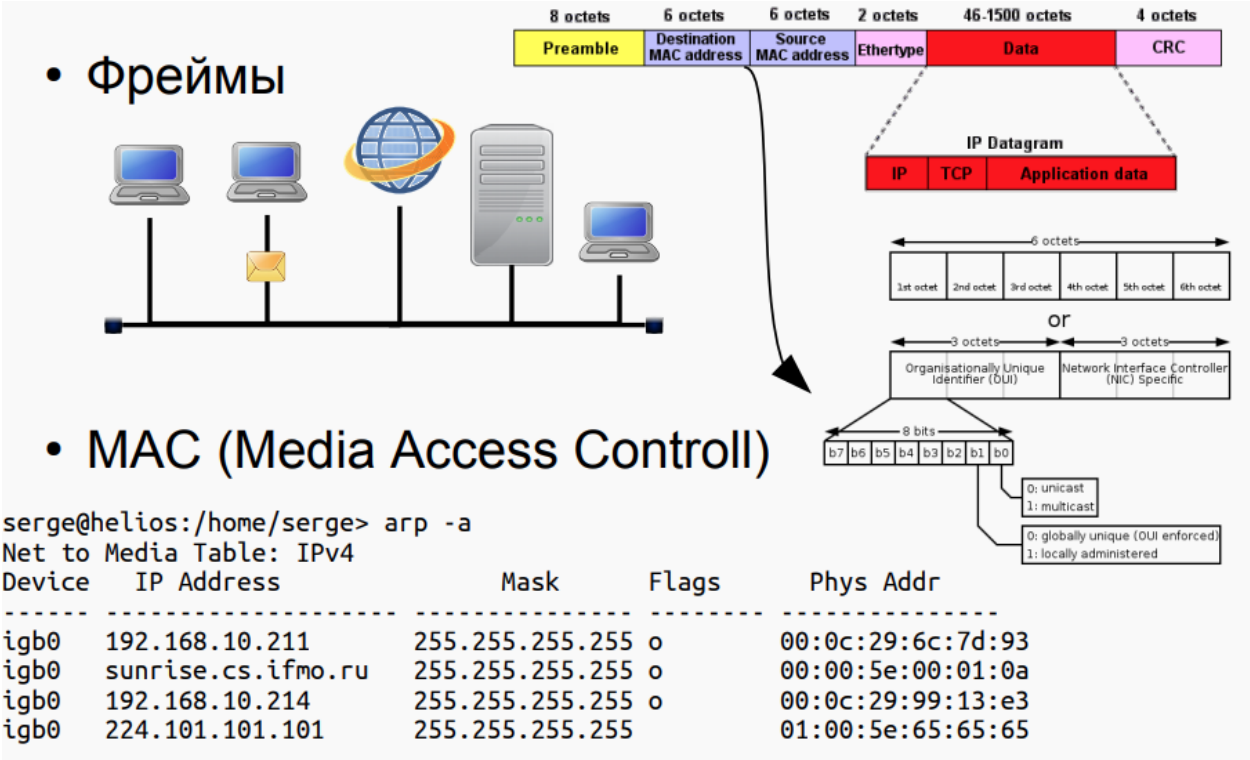
– ~500 м (Multi-mode fiber), ~80км (Single Mode)

● Wireless (802.11 - WiFi, 802.16 - WiMAX, 3G, 4G)

– 2.4, 5, 60 GHZ

– До 15 Гбит/с

Канальный уровень Ethernet:




```
serge@helios:/home/serge> arp -a
Net to Media Table: IPv4
Device   IP Address               Mask      Flags      Phys Addr
------ -------------------- --------------- -------- ---------------
igb0   192.168.10.211       255.255.255.255 o        00:0c:29:6c:7d:93
igb0   sunrise.cs.ifmo.ru   255.255.255.255 o        00:00:5e:00:01:0a
igb0   192.168.10.214       255.255.255.255 o        00:0c:29:99:13:e3
igb0   224.101.101.101      255.255.255.255          01:00:5e:65:65:65
```

Ethernet фрейм определяет формат сообщения данных, передаваемых по сети. Формат сообщения содержит несколько полей информации, в том числе данных, подлежащих передаче. Блок данных определяется как фактические данные, которые будут отправлены, и может содержать от 46 до 1500 байт двоичной информации. Длина блока данных определяется и включается в сообщении в качестве поля для приемника, чтобы определить, какая часть сообщения представляет собой данные.

MAC-адрес является шести байтовым двоичным номером набора, который включает в себя информацию источника и назначения для узлов. MAC-адрес включен в каждое сообщение и передается через сеть, а каждый узел сети Ethernet имеет уникальный MAC-адрес. MAC-адрес (Media Access Control —

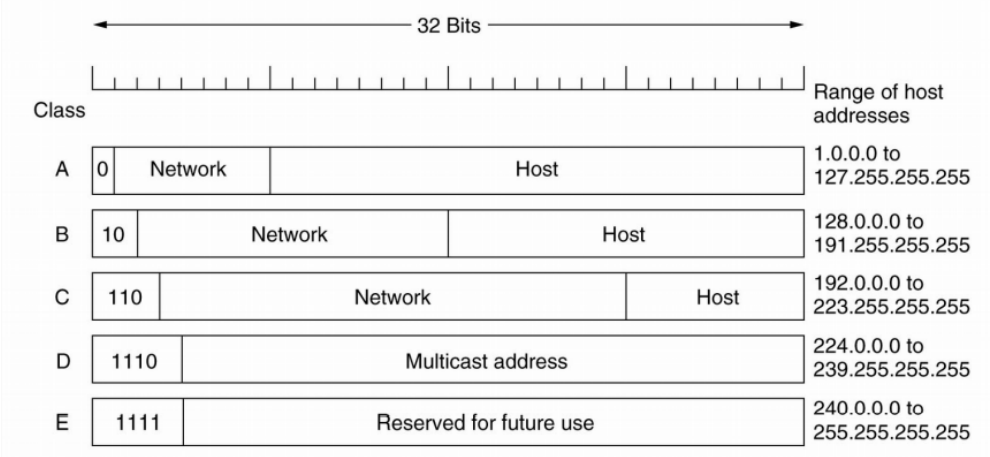
управление доступом к носителю) — это уникальный идентификатор, присваиваемый каждой единице оборудования компьютерных сетей.

Сетевой уровень предназначендля доставки пакетов от источника в пункт назначения, вероятно через множество физических сетей (линий связи). Сетевой уровень гарантирует, что каждый пакет будет доставлен от его исходной точки к его конечному пункту назначения. Некоторые обязанности сетевого уровня включают логическую адресацию и маршрутизацию.

Сетевой уровень предназначен для доставки отдельных пакетов от хоста источника до хоста пункта назначения.

- **Internet-протокол (IP)**. IP - механизм передачи, используемый в соответствии с протоколами TCP/IP. Это ненадежное без установления соединения обслуживание с *лучшими намерениями*..Термин " *с лучшими намерениями".* означает, что IP не обеспечивает проверки ошибок или выбор оптимального маршрута. IP учитывает ненадежность основных уровней и "прилагает все усилия", чтобы передать информацию к пункту назначения, но без гарантий. IP транспортирует данные в пакетах, называемых *дейтаграммами*, каждая из которых транспортируется отдельно. Дейтаграммы могут перемещаться по различным маршрутам и могут прибыть не в исходной последовательности или оказаться продублированными. IP не сохраняет порядок и список маршрутов и не имеет никаких средств для того, чтобы исправить дейтаграммы, однажды прибывшие в пункт назначения. Ограниченные функциональные возможности IP нельзя рассматривать как слабость. IP обеспечивает только функции передачи. Пользователь может добавить те средства, которые необходимы для данного приложения, и таким образом обеспечить максимальную эффективность.

• Адресация в IPv4 сетях



Маршрутизация (Routing) — это процесс по определению/вычислению лучшего маршрута движения для данных в сетях связи. Есть еще второе определение — это передача пакетов данных от отправителя к получателю.

Функцию роутинга могут выполнять:

- Аппаратные средства — маршрутизаторы. Самый оптимальный вариант, позволяющий обрабатывать большие потоки данных и работает он быстрее.
- Настроенные компьютеры с несколькими сетевыми интерфейсами и установленным на них специализированным и настроенным ПО. Обычно используется если конфигурация будет не слишком сложная.

Таблица маршрутизации — это файл-электронная таблица или база данных, которая располагается на маршрутизаторе или специально настроенном компьютере. В ней описывается соответствие адресов назначения с интерфейсами, через которые необходимо производить отправку данных до следующего маршрутизатора.

Таблица содержит:

- **Адрес** сети или узла

- **Маску подсети** назначения
- **Сетевой шлюз** или по-другому, адрес маршрутизатора на который будут направлены данные
- **Интерфейс**, с которого можно достучаться до шлюза
- **Метрика** (не всегда), т.е. показатель, который задает предпочтительность пути.

Может заполняться как вручную, так и автоматически.

33.Модель TCP/IP: выделение адресов (DHCP), сервисы имен, транспортный и прикладной уровни.

DHCP — протокол автоматизации назначения IP-адреса клиенту. Он широко используется в современных сетях. В статье рассмотрим принципы работы, процесс DORA, основные опции и другие аспекты протокола. DHCP — протокол прикладного уровня модели TCP/IP, служит для назначения IP-адреса клиенту. Это следует из его названия — Dynamic Host Configuration Protocol. IP-адрес можно назначать вручную каждому клиенту, то есть компьютеру в локальной сети. Но в больших сетях это очень трудозатратно, к тому же, чем больше локальная сеть, тем выше возрастает вероятность ошибки при настройке. Поэтому для автоматизации назначения IP был создан протокол DHCP.

Принцип работы DHCP

Из вступления ясно, какие функции предоставляет DHCP, но по какому принципу он работает? Получение адреса проходит в четыре шага. Этот процесс называют DORA по первым буквам каждого шага: Discovery, Offer, Request, Acknowledgement.

Discovery, или поиск

Изначально клиент находится в состоянии инициализации (INIT) и не имеет своего IP-адреса. Поэтому он отправляет широковещательное (broadcast) сообщение DHCPDISCOVER на все устройства в локальной сети. В той же локальной сети находится DHCP-сервер. DHCP-сервер — это, например, маршрутизатор или коммутатор, существуют также выделенные DHCP-серверы.

Offer, или предложение

DHCP-сервер отвечает на поиск предложением, он сообщает IP, который может подойти клиенту. IP выделяются из области (SCOPE) доступных адресов, которая задается администратором.

Request, или запрос

Клиент получает DHCPOFFER, а затем отправляет на сервер сообщение DHCPREQUEST. Этим сообщением он принимает предлагаемый адрес и уведомляет DHCP-сервер об этом. Широковещательное сообщение почти полностью дублирует DHCPDISCOVER, но содержит в себе уникальный IP, выделенный сервером. Таким образом, клиент сообщает всем доступным DHCP-серверам «да, я беру этот адрес», а сервера помечают IP как занятый.

Acknowledgement, или подтверждение

Сервер получает от клиента DHCPREQUEST и окончательно подтверждает передачу IP-адреса клиенту сообщением DHCPACK. Это широковещательное или

прямое сообщение утверждает не только владельца IP, но и срок, в течение которого клиент может использовать этот адрес.

Система доменных имен (DNS) является одной из фундаментальных технологий современной интернет-среды и представляет собой распределенную систему хранения и обработки информации о доменных зонах. Она необходима, в первую очередь, для соотнесения IP-адресов устройств в сети и более удобных для человеческого восприятия символьных имен.

DNS состоит из распределенной базы имен, чья структура напоминает логическое дерево, называемое пространством имен домена. Каждый узел в этом пространстве имеет свое уникальное имя. Это логическое дерево «растет» из корневого домена, который является самым верхним уровнем иерархии DNS и обозначается символом – точкой. А уже от корневого элемента ответвляются поддоменые зоны или узлы (компьютеры).

Сопоставление имен

Давайте взглянем, как происходит сопоставление имен и IP-адресов. Предположим, пользователь набирает в строке браузера www.1cloud.ru и нажимает Enter. Браузер посылает запрос DNS-серверу сети, а сервер, в свою очередь, либо отвечает сам (если ответ ему известен), либо пересылает запрос одному из высокоуровневых доменных серверов (или корневому).

Затем запрос начинает свое путешествие – корневой сервер пересылает его серверу первого уровня (поддерживающего зону .ru). Тот – серверу второго уровня (1cloud) и так далее, пока не найдется сервер, который точно знает запрошенное имя и адрес, либо знает, что такого имени не существует. После этого запрос начинает движение обратно. Чтобы наглядно объяснить, как это работает, ребята из dnssimple подготовили красочный комикс, который вы можете найти по ссылке.

Также стоит пару слов сказать про процедуру обратного сопоставления – получение имени по предоставленному IP-адресу. Это происходит, например, при проверках сервера электронной почты. Существует специальный домен in-addr.arpa, записи в котором используются для преобразования IP-адресов в символьные имена. Например, для получения DNS-имени для адреса 11.22.33.44 можно запросить у DNS-сервера запись 44.33.22.11.in-addr.arpa, и тот вернёт соответствующее символьное имя.

Транспортный уровень:

● TCP — Transmission Control Protocol

– Надежный, управляет перепосылкой данных

– Организует виртуальное соединение между гнездами (Socket=IP:port) на двух системах

– HTTP, FTP, SSH, SMTP, ...

● UDP — User Datagramm Protocol

– Послал сообщение и забыл

– Контроль надежности оставлен разработику

– Максимальная скорость передачи

– SNMP, TFTP, DHCP, DNS, ...

● Протоколы разрабатывает программист

```
public static void main (String args[]) throws Exception {
URL u = new URL("https://se.ifmo.ru/documents");
URLConnection c = u.openConnection();
BufferedReader s = new BufferedReader( new InputStreamReader(c.getInputStream()));
String line = null;
while ((line = s.readLine())!=null) {
System.out.println(line);
} }
```

34. Интерфейсы ввода-вывода. Контроллеры внешних устройств. Уровни стандартизации, сопряжения с системной шиной, циклы обмена. Регистры контроллера.

Интерфейсы:

● Определяет конкретные детали обмена

– Частота, набор каналов передачи, способ кодирования, команды, представления данных, набор данных и последовательность, ….

● Аппаратная и/или программная реализация

● Нуждаются в точной спецификации и/или стандартизации

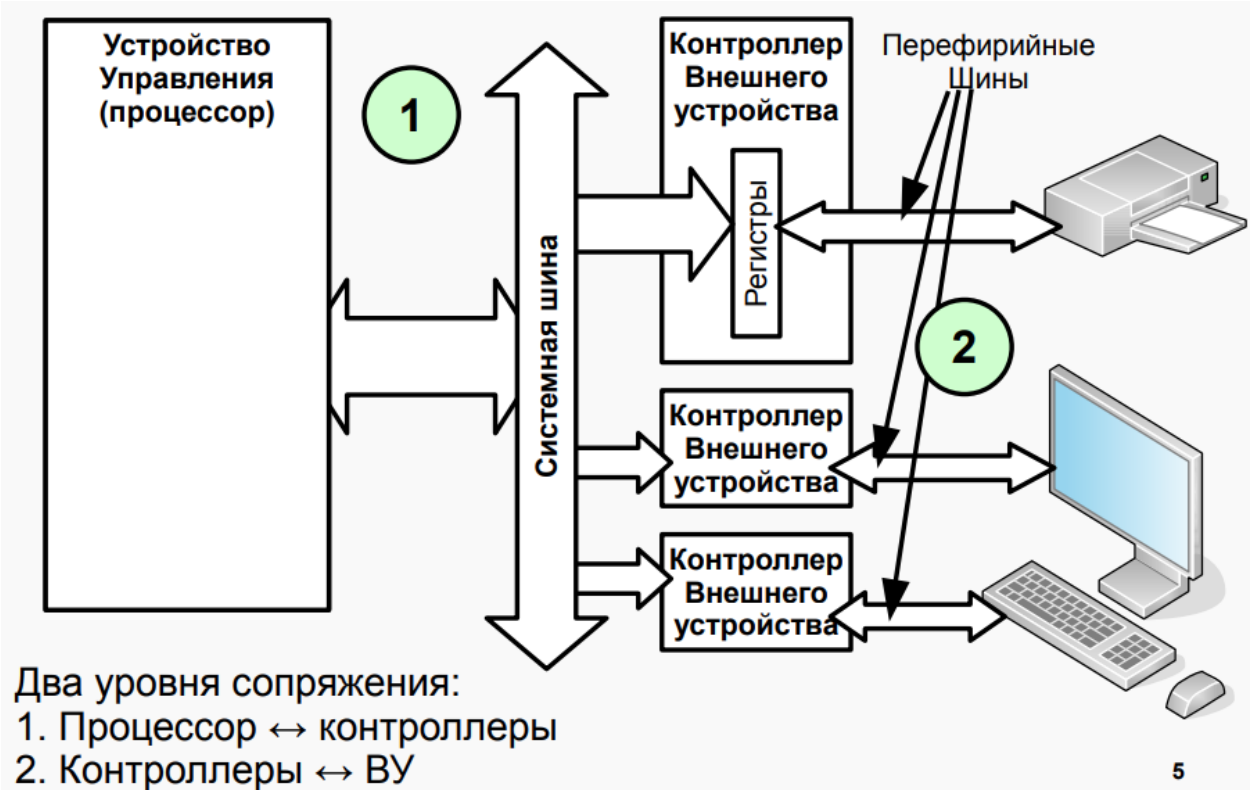
– Стороны обмена должны однозначно интерпретировать детали обмена

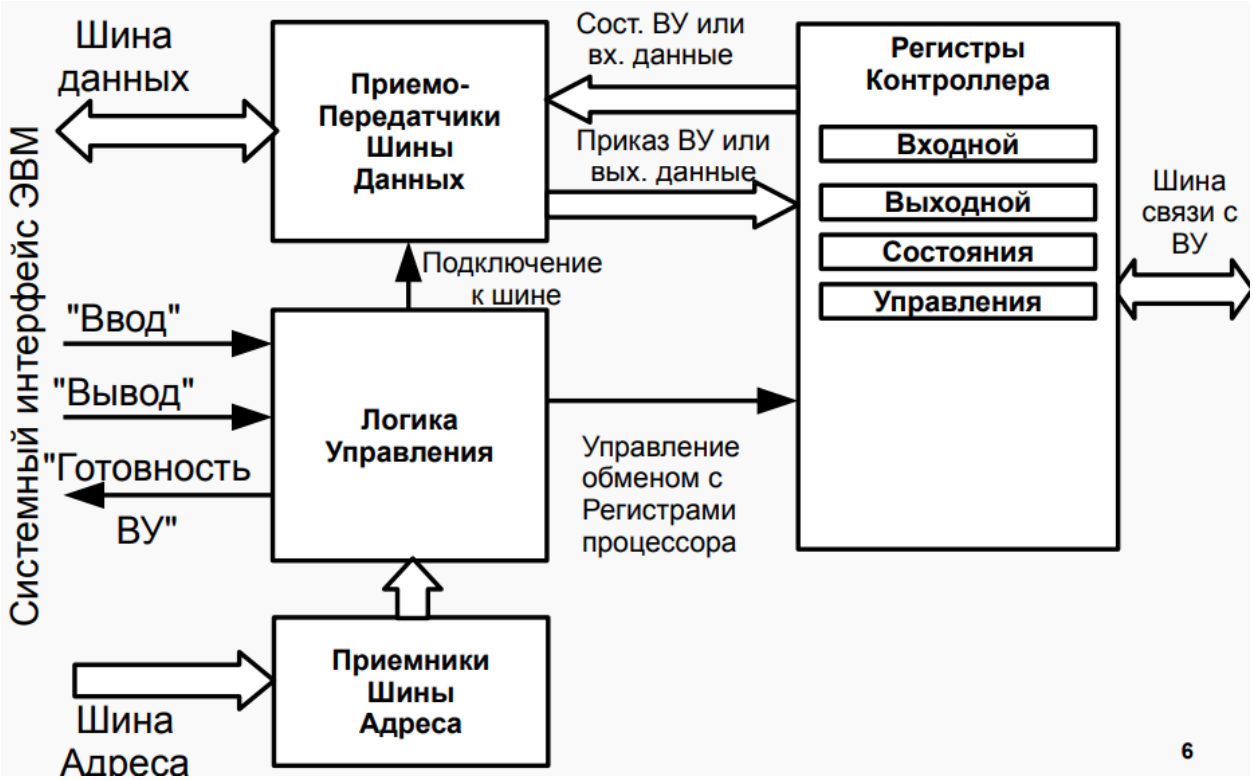
Уровни стандартизации интерфейсов

● Логическое подключение

● Физические параметры сигналов

● Конструктивные особенности





Контроллеры внешних устройств

Подключение любого внешнего устройства к микроЭВМ осуществляется через контроллер ВУ. Способы организации контроллеров определяются двумя факторами:

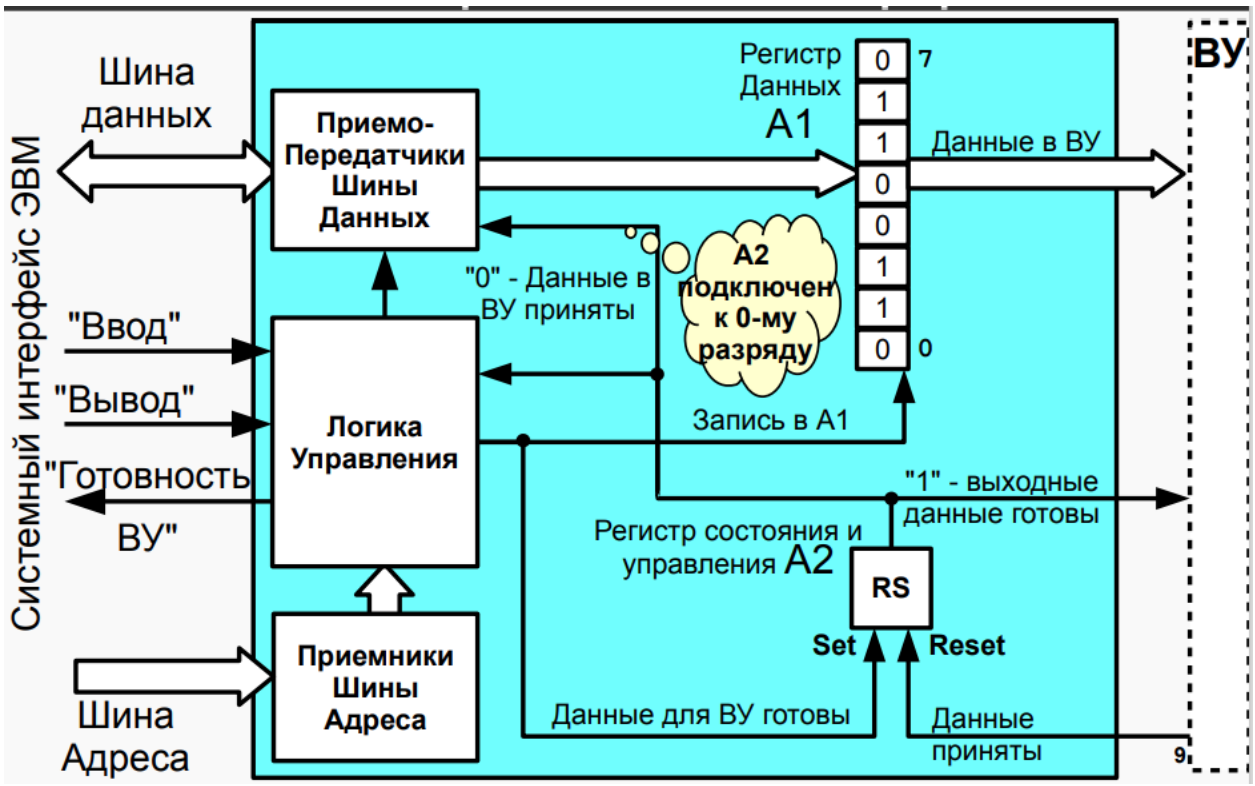
1. Форматами данных и режимами работы конкретных ВУ (существование самых разнообразных контроллеров от простейших до очень сложных)
2. Типом системного интерфейса микроЭВМ (определяет способ организации электронных схем контроллеров ВУ, обеспечивающих связь с шинами интерфейса, в первую очередь - схем распознавания адресов ВУ)

Для разных типов микроЭВМ разработаны контроллеры, обеспечивающие:

- Связь с ВУ по стандартному параллельному (ИРПР) каналу передачи данных;
- Связь с ВУ по стандартному последовательному (ИРПС) каналу передачи данных;
- Преобразование информации из аналоговой в цифровую с заданной точностью;
- Преобразование информации из дискретной формы представления в аналоговую в заданных диапазонах изменения аналоговых величин.

Существуют также программируемые контроллеры, режимы работы которых устанавливаются специальными командами микроЭВМ или определяются программами обмена с ВУ. Такие контроллеры необходимо настраивать на конкретный режим обмена (синхронный, асинхронный, с сигналами прерывания или без них). Настройка контроллеров производится программным путем с помощью спец. команд.

35. Параллельная передача данных. Контроллеры параллельной передачи и приема.



Параллельная передача данных в ВУ под управлением программы асинхронного обмена:

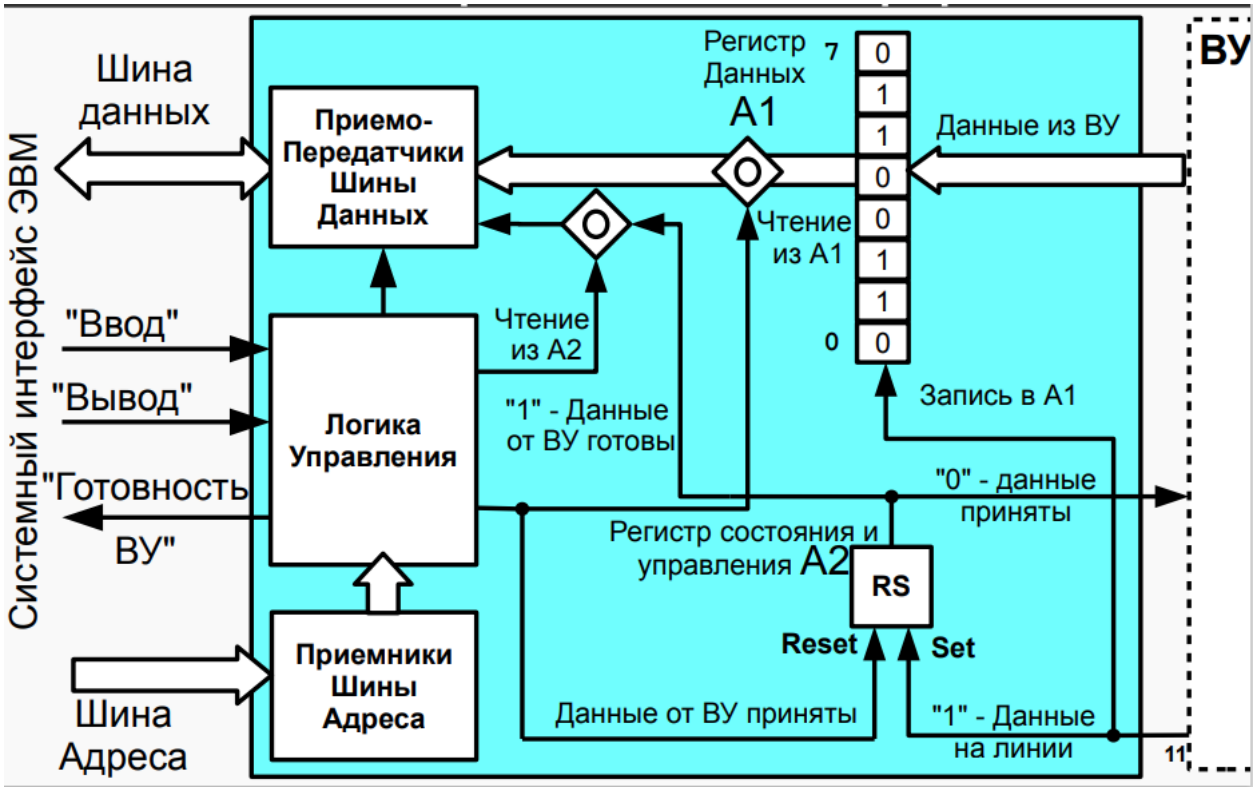
1. Процессор микроЭВМ проверяет готовность ВУ к приему данных

2. Если ВУ готово к приему данных (логический 0 в регистре А2), то данные передаются из шины данных системного интерфейса в регистр данных А1 контроллера и далее в ВУ. Иначе повторяется пункт 1.

В шине связи с ВУ используются 2 управляющих сигнала. Для формирования управляющего сигнала «Выходные данные готовы» и приема из ВУ управ. сигнала «Данные приняты» в контроллере используется одноразрядный адресуемый регистр состояния и управления А2. Одновременно с записью очередного байта данных из шины данных сист. интерфейса в адресуемый регистр данных контроллера А1 в регистр состояния и управления записывается логическая единица (формируется управляющий сигнал «Выходные данные готовы»).

ВУ, приняв байт данных, управ. сигналом «Данные приняты» обнуляет регистр состояния. При этом формируется:

1. Управ. сигнал сист. интерфейса «Готовность ВУ»
2. Признак готовности ВУ к обмену, передаваемый в процессор по одной из линий шины данных



Алгоритм асинхронного ввода:

1. Процессор проверяет наличие данных в регистре данных контроллера А1
2. Если данные готовы (логическая 1 в регистре А2), то они передаются из регистра данных А1 в шину данных системного интерфейса и далее в регистр процессора или ячейку памяти микроЭВМ. Иначе повторяется пункт 1.

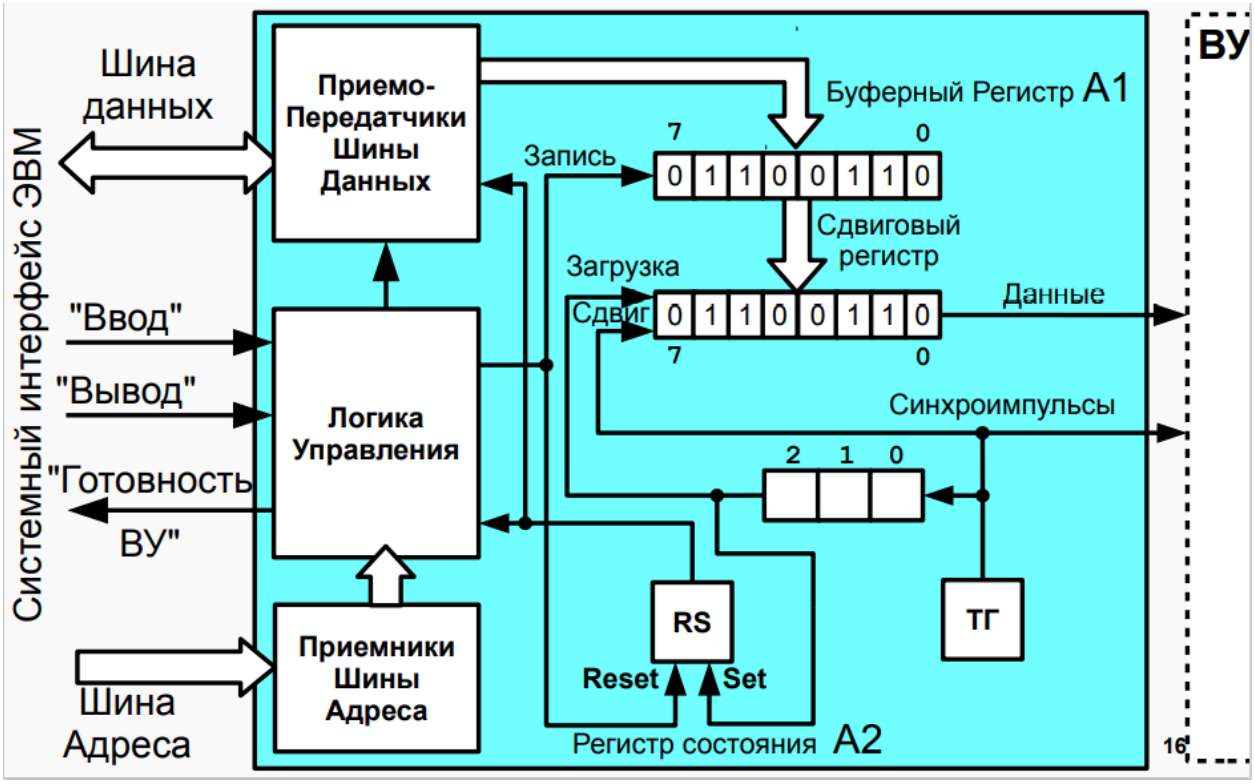
Для формирования управляющего сигнала «Данные приняты» и приема из ВУ управ. сигнала «Данные от ВУ готовы» в контроллере используется одноразрядный адресуемый регистр состояния и управления А2.

ВУ записывает в регистр данных контроллера А1 очередной байт данных и управ. сигналом «Данные от ВУ готовы» устанавливает в единицу регистр состояния и управления А2. При этом формируется:

1. Управ. сигнал сист. интерфейса «Готовность ВУ»
2. Признак готовности ВУ к обмену, передаваемый в процессор по одной из линий шины данных

Так контроллер извещает процессор о готовности данных в регистре А1. Процессор читает байт данных из регистра данных контроллера и обнуляет регистр состояния и управления А2. При этом формируется управляющий сигнал «Данные приняты».

36. Синхронные последовательные интерфейсы. Контроллеры последовательной передачи и приема.

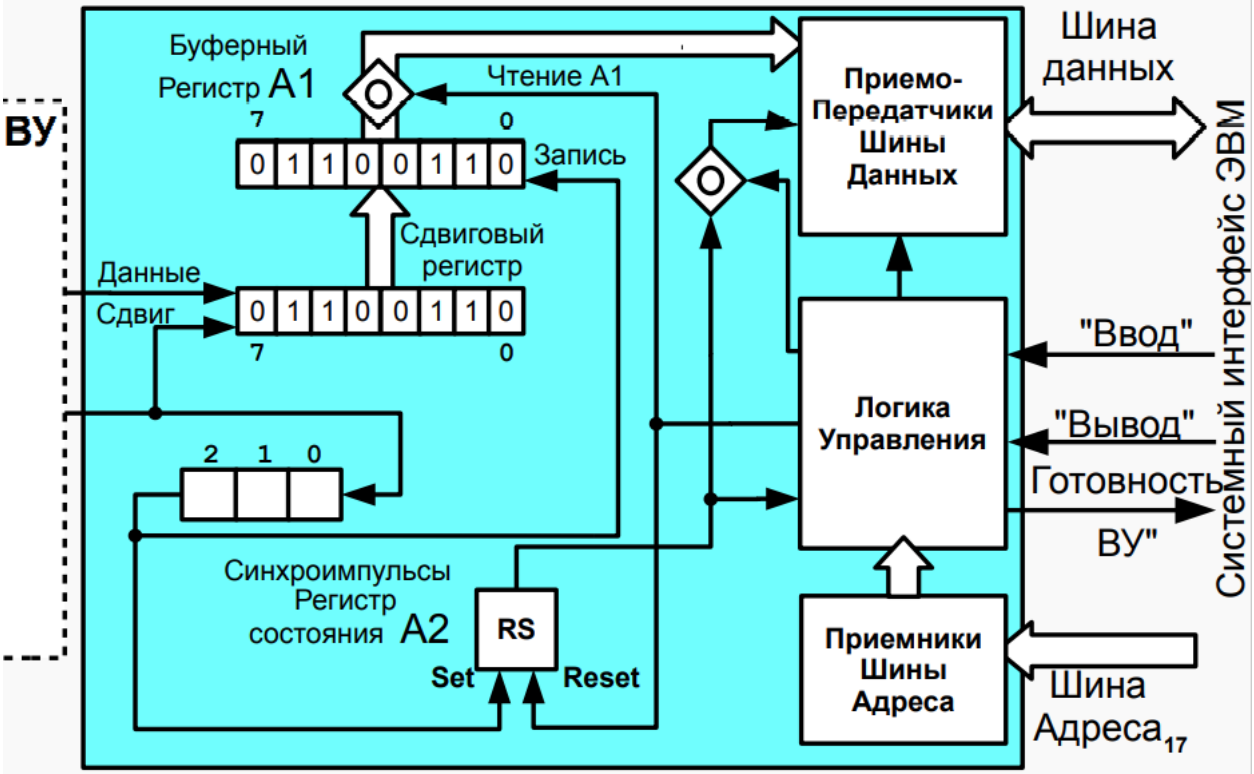


8-ми разрядный буферный регистр контроллера А1 - для временного хранения байта данных до его загрузки в сдвиговый регистр. Запись байта данных в буферный регистр происходит при наличии 1 в регистре состояния А2. Содержимое этого регистра передается в процессор по одной из линий шины данных и используется для формирования управ. сигнала «Готовность ВУ». При записи очередного байта в регистр А1 обнуляется регистр А2.

В сдвиговом регистре происходит преобразование данных из параллельного формата в последовательный и передача их в линию связи. По очередному тактовому импульсу содержимое

сдвигового регистра сдвигается на 1 разряд вправо и в линию связи «Данные» выдается значение очередного разряда. Одновременно со сдвигом по линии «Синхронизация» передается тактовый импульс.

Количество переданных в линию тактовых сигналов (переданных бит) подсчитывается счетчиком тактовых импульсов. Как только его содержимое равно 7(передано 8 бит информации) формируется управляющий сигнал «Загрузка» и происходит запись в сдвиговый регистр очередного байта из буферного. Устанавливается в 1 регистр состояния. Следующим тактовым импульсом счетчик будет сброшен в 0 и начнется очередной цикл выдачи 8 бит из сдвигового регистра в линию связи.



Буферный регистр контроллера A1 - для временного хранения байта , поступившего из сдвигового регистра. Чтение байта данных из буферного регистра происходит при наличии 1 в регистре состояния A2.

Данные, поступающие из линии связи в последовательном коде преобразуются в параллельный с помощью сдвигового регистра. Линия «Данные» подключается в контроллере к последовательному входу сдвигового регистра, а линия «Синхронизация» - на управ. вход «Сдвиг» и на вход счетчика тактовых импульсов. По тактовому импульсу по линии «Синхронизация» производится сдвиг содержимого сдвигового регистра на 1 разряд влево и запись очередного бита данных из линии «Данные» в младший разряд этого регистра. Одновременно увеличивается на 1 счетчик тактовых импульсов. Как только он становится равным 7 формируется управ. сигнал «Запись» и происходит запись в буферный регистр байта из сдвигового. Устанавливается в 1 регистр состояния.

При передаче байта данных из буферного регистра в шину данных регистр состояния обнуляется (т.е. в сдвиговый регистр принимается очередной байт информации).

37.Асинхронный обмен. Принципы деления частоты, формат кадра.

Асинхронный обмен

При реализации *асинхронного обмена* интервал между командами передачи данных задается самим внешним устройством. Контроллеры этих устройств снабжаются регистром состояния, который информирует ЭВМ о готовности устройства к обмену информацией.

Обмен происходит по готовности ВУ, из этого вытекают следующие:

Преимущества:

1. Не нужно знать время выполнения операции на ВУ;
2. ВУ всегда успеет выполнить обработку данных перед началом следующей операции обмена.

Недостаток:

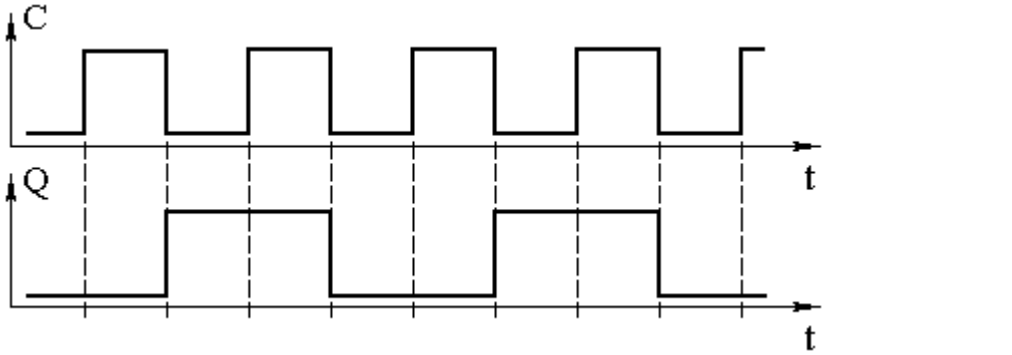
1. ЭВМ не выполняет никаких полезных действий во время ожидания момента готовности ВУ.

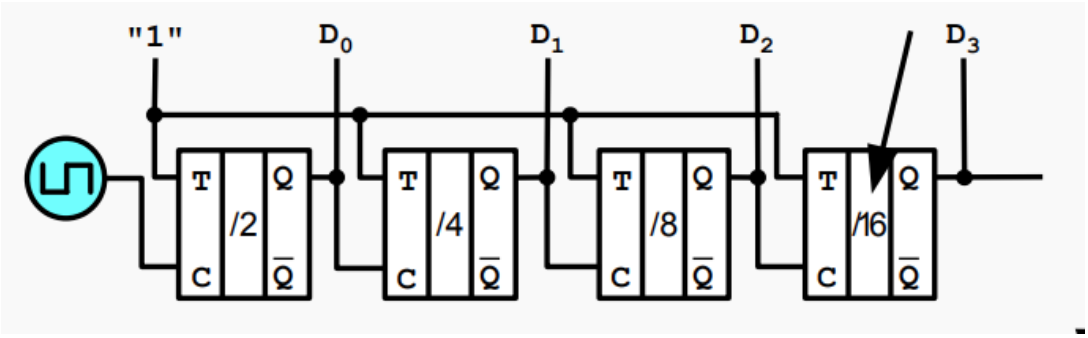
Принципы деления частоты

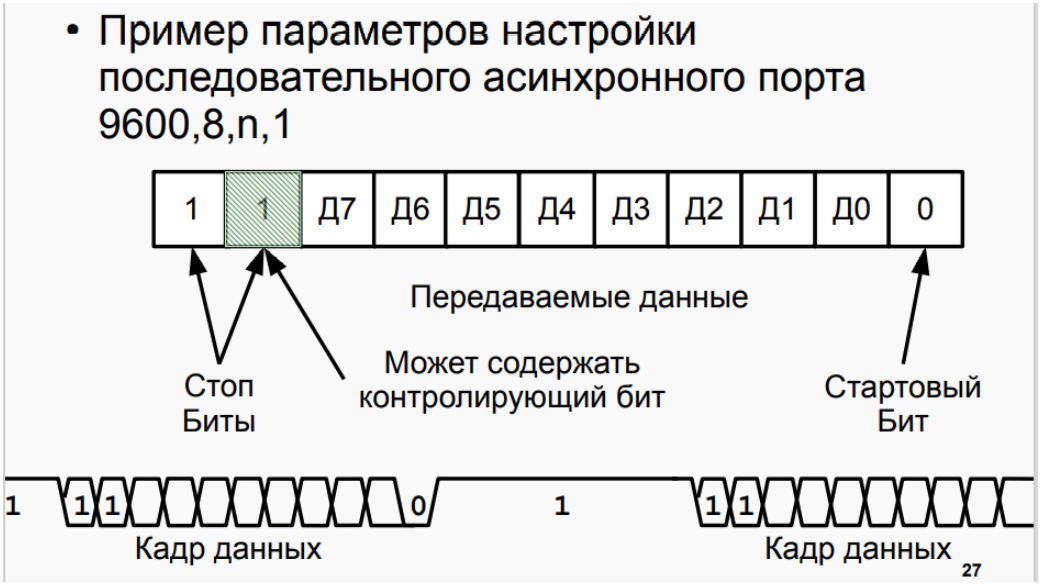
При асинхронной передаче данных, со временем может происходить рассинхронизация генераторов тактов передатчика и приёмника, в результате чего данные могут быть переданы с искажениями, или не быть переданы вовсе.

Одним из способов решения этой проблемы является деление тактовой частоты генераторов.

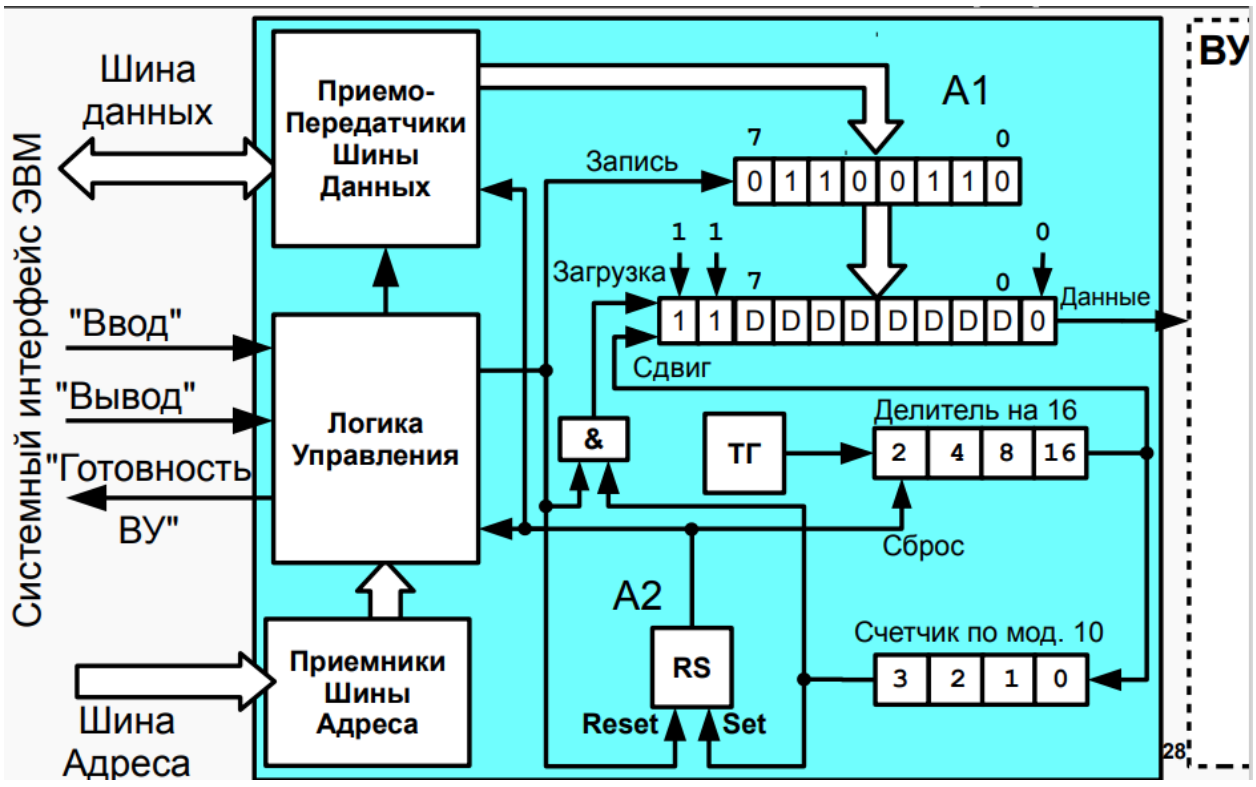
Принцип его заключается в том, что исходная тактовая частота делится с некоторым коэффициентом, чаще всего кратным степени двойки. Таким образом увеличивается область совпадения фаз и время передачи данных, что увеличивает точность передачи.







38. Контроллер передачи асинхронного последовательные интерфейса.

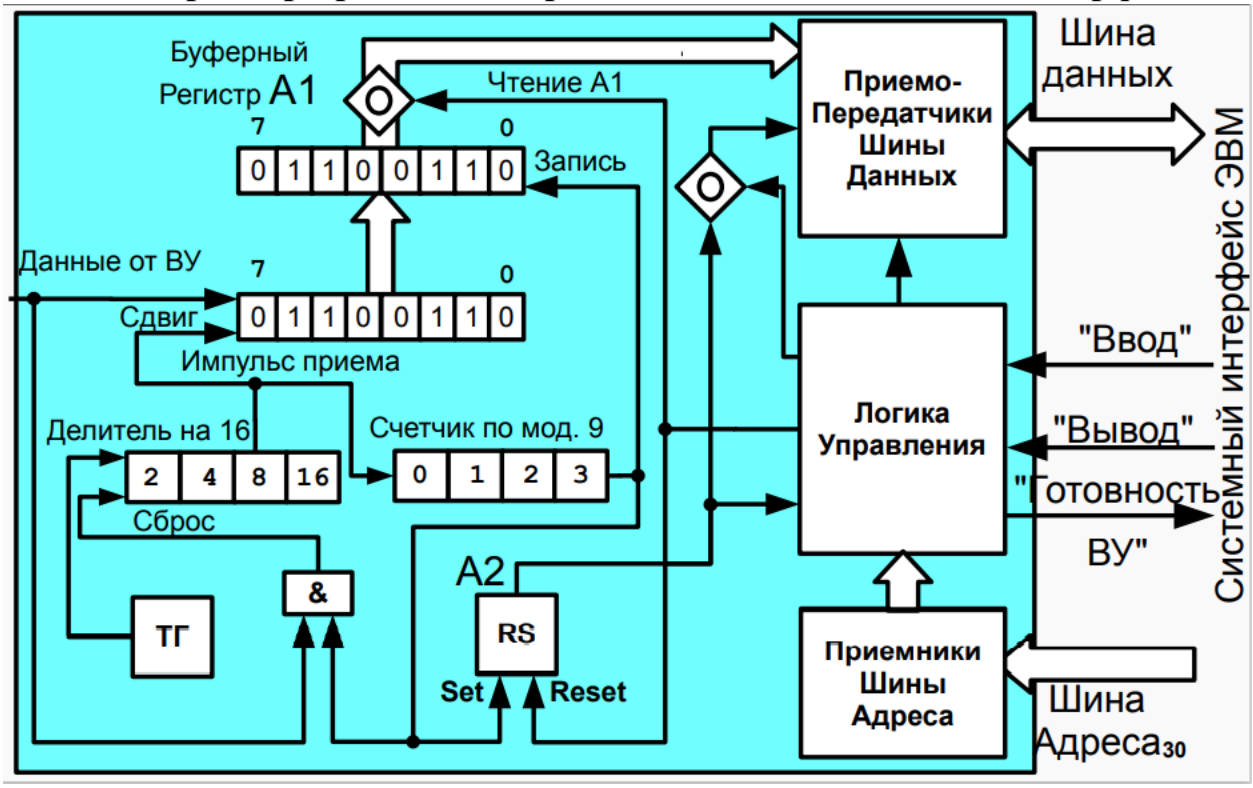


Процесс передачи

1. После передачи предыдущих байтов данных в Регистр Состояния А2 записывается 1, что информирует процессор о готовности контроллера к приему следующего байта данных и передаче его по линии связи в ВУ. Он же запрещает формирование импульсов со схемы выработки импульсов сдвига – делителя частоты тактового генератора на 16 (счетчик по mod 16). Счетчик импульсов сдвига (счетчик по mod 10) находится в нулевом состоянии, и его единичный выходной сигнал поступает на вентиль И, подготавливая цепь выработки сигнала загрузки сдвигового регистра.

2. Процессор, выполняя команду «Вывод», выставляет передаваемый байт на шине данных и формирует управляющий сигнал системного интерфейса «Вывод».
3. По сигналу «Вывод» в контроллере происходит запись передаваемого байта в буферный регистр А1, сброс регистра состояния А2 и формирование на вентиле И сигнала «Загрузка».
4. Передаваемый бит переписывается в разряды 1..8 сдвигового регистра, в 0 разряд записывается нуль – стартовый бит, а в разряды 9 и 10 единицы – стоповые биты.
5. Снимается сигнал «Сброс» с делителя частоты, он начинает накапливать импульсы генератора тактовой частоты и в момент приема шестнадцатого тактового импульса срабатывает импульс сдвига (так реализовано деление частоты).
6. На шине «Данные» поддерживается 0 (значение стартового бита) до тех пор, пока не будет выработан первый импульс сдвига (время передачи 1 бита). Импульс сдвига изменит состояние счетчика импульсов сдвига и перепишет в нулевой разряд сдвигового регистра первый информационный бит передаваемого байта данных. Значение этого бита будет поддерживаться на линии «Данные» до следующего импульса сдвига.
7. Аналогично передаются остальные информационные биты, первый стоповый бит, и, наконец второй стоповый бит, при передаче которого счетчик импульсов сдвига снова установится в нулевое состояние. Это приведет к записи 1 в регистр состояния А2. Единичный сигнал с выхода регистра А2 запретит формирование импульсов сдвига, и информирует о готовности к приему нового байта данных.
8. После завершения передачи очередного кадра (стартового бита, информационного бита и 2х- стоповых битов), на линии передачи данных поддерживается значение второго стопового бита – единицы

39.Контроллер приема асинхронного последовательные интерфейса.



Процесс передачи

1. На линии «Данные» находится единица, что запрещает работу делителя частоты генератора тактовых импульсов.

2. При обнаружении нулевого сигнала на линии «Данные» (смена стопового бита на стартовый), снимается сигнал «Сброс» с делителя частоты, он начинает накапливать импульсы генератора тактовой частоты.
3. Когда на счетчике накопится значение 8 (половина времени передачи бита), он выдаст сигнал, поступающий на входы сдвигового регистра и счетчика импульсов сдвига. (Таким образом уменьшается вероятность искажения данных.)
4. Последующие сдвиги происходят после прохождения 16-ти тактовых импульсов.
5. При приеме в сдвиговый регистр 9-го бита кадра (8-го инф. Бита), из него выдвинется стартовый бит, и, следовательно в сдвиговом регистра будет размещен информационный байт. В этот момент счетчик импульсов сдвига придет в нулевое состояние и на его выходе будет выработан единичный сигнал, по которому:
    a. Содержимое сдвигового регистра будет переписано в БР
    b. В РС А2 запишется 1 и он будет информировать процессор об окончании приема очередного байта
    c. Вентиль И подготовится к вы работке сигнала «Сброс»(он сформируется после прихода первого стопового бита).
    d. Получив сигнал готовности (1 в РС А2), процессор выполнит команду «Ввод». При этом вырабатывается сигнал системного интерфейса «Ввод», по которому производится пересылка принятого байта данных из БР в процессор (сигнал «Чтение») и сброс РС А2

40.Организация прямого доступа к памяти. Контроллер ПДП

Этот тип обмена реализуется полностью аппаратно и управляется контроллером ПДП.

Организация ПДП

Особенности ПДП

1. Возможность начальной загрузки программ в основную память микроЭВМ из устройства ввода.
2. Обеспечивает возможность использования в микроЭВМ быстродействующих внешних запоминающих у-в.

Для экономии ресурсов контроллер ПДП не имеет свои ресурсы, а подключается к шинам данных (ШД) и адреса (ША) системного интерфейса ЭВМ, что делает невозможным одновременное использование шин контроллером ПДП и процессором.

Эта проблема решается двумя способами:

1. Захват цикла
    a. Простой захват цикла.
    Передача происходит в те машинные циклы, в которых процессор не обменивается данными с памятью. Пометка таких циклов выполняется либо с помощью спец. указывающего цикла, либо такие циклы выбираются с помощью спец. селектирующих схем, что усложняет конструкцию процессоров.
    При таком способе организации обмена ПДП не снижает производительности микроЭВМ, но обмен возможно только в случайные моменты времени одиночными байтами или словами.

b. Захват цикла с принудительным отключением процессора от шин системного интерфейса.
   Для его реализации такого режима ПДП системный интерфейс (СИ) дополняется двумя линиями для передачи управляющих сигналов «Требование ПДП» (ТПДП) и «Предоставление ПДП» (ППДП).

1. ТПДП формируется контроллером ПДП.
2. Получив сигнал ТПДП, процессор приостанавливает выполнение очередной команды, не дожидаясь её завершения, выдает в системный интерфейс ППДП и отключается от шин СИ
3. Контроллер ПДП получает управления над шинами СИ и осуществляет обмен одним байтом или словом данных с памятью микроЭВМ.
4. Контроллер ПДП возвращает управление СИ процессору.

2. Блокировка процессора
   Отличается от «Захвата цикла» тем, что управление СИ передается контроллеру ПДП не на время обмена одним байтом, а на время обмена блоком данных.

Контроллер ПДП ввода данных из ВУ в режиме «Захват цикла»

1. Процессор загружает в СК контроллера количество принимаемых байтов, а в РА контроллера начальный адрес области памяти для вводимых данных.
2. Байты данных из ВУ поступают в РД контроллера, при этом каждый байт сопровождается управляющим сигналом из ВУ «Ввод данных», который обеспечивает запись байта в РД контроллера. По этому же сигналу при ненулевом состоянии СК контроллер формирует сигнал ТПДП.
3. По ответному сигналу процессора ППДП контроллер выставляет на ША и ЩД содержимое своих РА и РД.

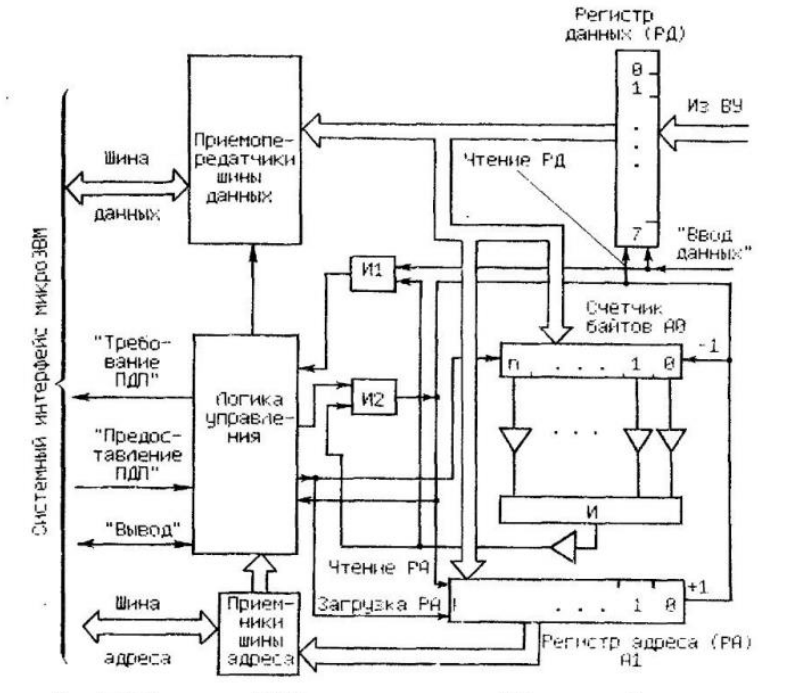


Рис. 8. 16. Контроллер ПДП для ввода данных из ВУ в режиме «Захват цикла»

4. Формируя приказ «Вывод», контроллер ПДП обеспечивает запись байта данных из своего регистра данных в память микроЭВМ.
5. По тому же сигналу ППДП содержимое СК декрементируется, а содержимое РА обновляется. Как только СК станет равным нулю, контроллер прекратит формирование сигналов ТПДП