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. 2](#_Toc74945125)

[2. Представление чисел с фиксированной точкой. Прямой, обратный и дополнительный код. Формирование битовых признаков переноса, переполнения, отрицательного результата, нуля. 4](#_Toc74945126)

[3. Представление символьных и строковых данных. Принципы построения кодовых таблиц ASCII, КОИ-8, ISO8859-5, Windows-1251, UTF-8, UTF-16. 5](#_Toc74945127)

[4. Базовые элементы вычислительной техники: ячейки, регистры, шины, вентили, тактовые генераторы, логические схемы, триггеры, регистры, счетчики, сумматоры. 7](#_Toc74945128)

[5. Структура и принцип функционирования ЭВМ. Порядок функционирования простого процессора на примере калькулятора. 9](#_Toc74945129)

[6. Операционная система Unix — ядро ОС и файловая система. 11](#_Toc74945130)

[7. Операционная система Unix — интерпретаторы, стандартные потоки ввода вывода, фильтры. 12](#_Toc74945131)

[8. Операционная система Unix — основные команды, права файлов и способы их задания. 13](#_Toc74945132)

[9. Состав и структура БЭВМ. Адресные пространства БЭВМ. Система команд БЭВМ, форматы команд. Машинные циклы. 15](#_Toc74945133)

[10. Организация вычислений в БЭВМ. Сдвиги, арифметические и логические операции. Цикл выборки команды. 17](#_Toc74945134)

[11. Организация массивов данных. Режимы адресации. Цикл выборки адреса и операнда БЭВМ. 18](#_Toc74945135)

[12. Управление вычислительным процессом в БЭВМ. Команды ветвлений, цикл исполнения команды LOOP. 19](#_Toc74945136)

[13. Подпрограммы в БЭВМ. Цикл исполнения команд перехода и возврата из подпрограммы. Стек, передача параметров. Позиционно-независимый код. Загрузчик и библиотеки. 20](#_Toc74945137)

[14. Организация ввода-вывода в вычислительных системах. Инициация обмена, передача информации и завершение обмена. Драйверы. 22](#_Toc74945138)

[15. Организация ввода-вывода в БЭВМ. Устройства ввода-вывода, команды. 22](#_Toc74945139)

[16. Организация асинхронного обмена в БЭВМ. Пример программы. Временные издержки асинхронного обмена. 23](#_Toc74945140)

[17. Организация прерываний в БЭВМ. Вектора прерываний, контроллер прерывания. 24](#_Toc74945141)

[18. Организация обмена по прерыванию программы в БЭВМ. Пример программы. Цикл прерывания. 26](#_Toc74945142)

[19. Понятие многоуровневой ЭВМ. Понятие и пример программы на разных уровнях. 27](#_Toc74945143)

[20. Микропрограммный уровень БЭВМ. Структура МПУ. Форматы микрокоманд. 28](#_Toc74945144)

[21. Структура и принципы работы арифметико-логического устройства и коммутатора. Регистр состояния БЭВМ 29](#_Toc74945145)

[22. Микропрограммное управление вентильными схемами. Схема управления. Интерпретатор БЭВМ. 32](#_Toc74945146)

[23. Архитектура ЭВМ. Гарвардская и фон-Неймановская архитектура. Организация обмена архитектуры ЭВМ с использованием шин. 33](#_Toc74945147)

[24. Архитектура многопроцессорных ЭВМ. Системный коммутатор. Архитектуры UMA и NUMA. 34](#_Toc74945148)

[25. Структура современных процессоров. Окружение процессора. CISC, RISC, VLIW. 35](#_Toc74945149)

[26. Адресуемая память, организация и временные диаграммы. Конструктивные особенности современной памяти. 37](#_Toc74945150)

[27. Память, ориентированная на записи (блочная память). Организация дисковой памяти и памяти на магнитных лентах. 38](#_Toc74945151)

[28. Характеристики запоминающих устройств. Пирамида памяти. 39](#_Toc74945152)

[29. Ассоциативная память, Кэш-память. Влияние промахов кэш-памяти на производительность. 40](#_Toc74945153)

[30. Предназначение и организация виртуальной памяти. Сегментно-страничная организация. Устройство управления памятью (MMU), буфер трансляции (TLB). 42](#_Toc74945154)

[31. Сетевые технологии, Понятие сети ЭВМ, классификация компьютерных сетей. Сообщение и пакет. Модель взаимодействия открытых систем. 44](#_Toc74945155)

[32. Модель TCP/IP: передающая среда, канальный и сетевой уровень. Адресация, передача и маршрутизация пакетов. 47](#_Toc74945156)

[33. Модель TCP/IP: выделение адресов (DHCP), сервисы имен, транспортный и прикладной уровни. 49](#_Toc74945157)

[34. Интерфейсы ввода-вывода. Контроллеры внешних устройств. Уровни стандартизации, сопряжения с системной шиной, циклы обмена. Регистры контроллера. 52](#_Toc74945158)

[35. Параллельная передача данных. Контроллеры параллельной передачи и приема. 54](#_Toc74945159)

[36. Синхронные последовательные интерфейсы. Контроллеры последовательной передачи и приема. 56](#_Toc74945160)

[37. Асинхронный обмен. Принципы деления частоты, формат кадра. 57](#_Toc74945161)

[38. Контроллер передачи асинхронного последовательные интерфейса. 59](#_Toc74945162)

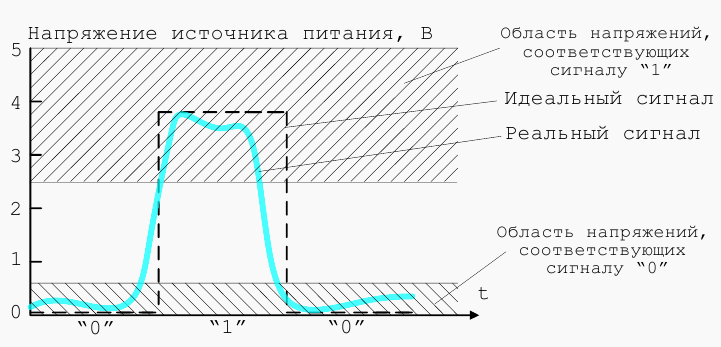
[39. Контроллер приема асинхронного последовательные интерфейса. 60](#_Toc74945163)

[40. Организация прямого доступа к памяти. Контроллер ПДП 61](#_Toc74945164)

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~~

第一种信息表示形式称为模拟或连续（借助类似值 - 模拟）。以这种形式表示的量可以采用的值数量是无限大的，即使大小在有限的范围内变化。因此得名——连续量级和连续信息。连续性这个词突出了这些量的主要属性——没有间隙，模拟量可以采用的值之间的间隔。一个值表示为与该值成正比的单个信号。这种表示形式用于模拟计算

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.

第二种信息表示形式称为数字或离散（借助一组电压，每个电压对应于所表示数量的一个数字）。这种值不接受所有可能的值，而只接受明确定义的值，称为离散（不连续）。与连续值不同，离散量的值数始终是有限的。该值以多个信号的形式表示，每个信号对应给定值的一个数字。这种表示形式用于电子计算机（计算机）。 每个值来自问题的初始数据集，其求解结果可以在计算机中以几个电信号的形式表示，其中一个对应于值中的单位数，另一个对应于十的数量，第三个对应于数百的数量，等等。设计用于处理此类信号的设备必须区分每个信号中的十个状态。构建一个仅区分两种状态（其存在或不存在）的设备要容易得多。这更加方便，因为目前现有的用于将数据输入计算机的廉价设备也使用两种状态对输入信息的各个组件进行编码。 这促使第一台计算机的创建者使用不同的数字系统来表示机器中的数字：采用二进制数系统代替通常的十进制数系统。2CC 也是一个位置 CC，即在其中，每个数字的值取决于该数字在数字符号中的位置。 计算中有一些特殊术语被广泛使用：位、字节和字。

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=23, 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)

二进制位 八位单位 – 字节 计算机包含大量存储单元和用于存储二进制信息的寄存器。这些单元中的大多数具有相同的长度 n，即它们用于存储 n 位二进制信息。存储在此类单元格中的信息称为单词。 二进制数系统，方便在电脑中使用，对于一个人写和读数字来说完全不方便。为了减少手动处理数字代码和命令的劳动强度，8CC和16CC被广泛使用。8 CC 使用 8 位数字 （0-7），16CC 使用 10 位数字和 6 个大写字母（0-9，AF）。由于8CC的基数是8=23，为了将二进制数转换为八进制数，需要将二进制数分成三元组。这些组中的每一个都可以用一个八进制数字表示。以同样的方式，将二进制数转换为十六进制数。只有在这种情况下，二进制数才被分成 4 个四分体，由一个十六进制数字表示。 最后，有必要提到二进制十进制 CC，它用于数字设备，其中作的主要部分与输入信息的处理和存储无关，而是与其输出到一些指标有关，并以十进制表示所获得的结果。在 2-10CC 中，从 0 到 9 的十进制数字表示从 0000 到 1001 的 4 位二进制组合。两个二进制十进制数字是 1 个字节（可以表示 0 到 99 之间的值）

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

定点数的表示。正向、反向和附加代码。形成转移位符号，溢出，负结果，零。

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 = 216 - 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.

无符号二进制整数可用于表示零整数和正整数。当此类数字放置在单个 16 位字中时，它们可以从 （0000 0000 0000 0000）2 = （0000）16 = 0 到 （1111 1111 1111 1111）2 = （FFFF）16 = 216 - 1 = 65535。这样的条目称为直接 数字代码。 相似的数字（以及下面讨论的带有符号的二进制数） 指用定点分隔整数和小数部分的数字 数。在基本计算机中使用的数字中，逗号的位置严格要求 在单词的低点之后固定。 必要时使用有符号二进制整数 区分正数和负数。在现代计算机中 有符号整数的表示使用附加代码，其中 格式的最有效位定义数字的符号：0 表示正数，1 表示 负数。同时，正数的附加码 与其直接代码一致。并表示 中的负数 附加代码反转数字的直接模码 （获取数字的反代码）并将 1 添加到结果中。一样 当您更改附加代码中表示的数字的符号时，将使用此作。 使用附加代码简化了计算机的设计，因为 将两个具有不同符号的此类数字相加时，您无需转到 从较大的数字中减去较小（模）数并赋值的运算 较高数字符号的结果。此外，相同的加法器电路 可用于对有符号和无符号执行作 数字的表示。

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

+

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 *（1）*

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

字符和字符串数据的表示。构建 ASCII、KOI-8、ISO8859-5、Windows-1251、UTF-8、UTF-16 代码表的原则。

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.

计算机中文本信息的表示是基于地球上存在的人类传统使用的语言的字母编码。符号是人们用来创建文字、文本和其他重要信息的图形图像。如您所知，符号包括字母、标点符号、货币符号、数字等。 在计算机中，文本信息的表示基于使用代码表对字母和符号进行编码。代码表（或字符编码）是开发人员之间的协议，将每个字符与特定的序列号或代码进行匹配，以便它可以存储在计算机内存中或通过通信通道传输。 有用于在计算机中存储符号图形样式的字体。字体可以是矢量字体和位图字体。在位图字体中，每个字符代码对应于特定大小（以点为单位）的图像。矢量模型存储以一系列线条的形式绘制符号的原理。

**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.

ASCII 是包含主要字符（英文字母、数字、标点符号、国家字母字符（每个地区不同）、服务字符）和每个字符的代码长度 n = 8 位的编码表。

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).

7 位： ASCII 是第一个适合处理文本的编码。除了英文字母和服务符号的小写字母外，它还包含英文的大写字母、数字、标点符号和其他符号（在这种情况下，最重要的位用于控制通过通信通道传输的符号位的奇偶校验）。

Кодировки стандарта ASCII (8 бит):

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

ISO 8859 是第一个可以使用国家字母字符的编码。

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).

KOI8-R 是第一个俄语编码。西里尔字母不按字母顺序排列。它们被放置在表格的上半部分，以便西里尔字母的位置与英文字母表中的语音对应物相对应。这意味着即使每个字符的最高有效位丢失，例如，当通过过时的七位调制解调器时，文本仍然“可读”（因此有音译的概念）。

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).

Windows-1251 是 90 年代初俄语版本的 Windows作系统中使用的俄语编码。西里尔字母按字母顺序排列。包含纯文本排版中的所有字符（重音符号除外）。

**Кодировки стандарта UNICODE**

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

Unicode 是一种行业标准，提供世界上所有脚本和特殊字符的字符的数字表示。

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.

该标准由非营利组织 Unicode Consortium 于 1991 年提出。使用此标准可以对来自不同脚本的大量字符进行编码。该标准由两个主要部分组成：通用字符集 （UCS） 和 Unicode 转换格式 （UTF）。通用字符集指定字符与代码的一对一对应关系，即表示非负整数的代码空间元素。编码系列定义了 UCS 代码序列的机器表示。

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-8 是一种 Unicode 表示形式，可与使用 8 位字符的旧系统提供最佳兼容性。当写入 UTF-8 时，仅由编号小于 128 的字符组成的文本将转换为纯 ASCII 文本。相反，在 UTF-8 文本中，任何值小于 128 的字节都表示具有相同代码的 ASCII 字符。其他 Unicode 字符以 2 到 6 个字节长的序列表示。

**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».

Unicode 的第一个版本（1991 年）是具有固定字符宽度的 16 位编码;不同符号的总数为 65,536 个。在 Unicode 的第二个版本（1996 年）中，决定大幅扩展代码区域;创建 UTF-16 是为了保持与那些已经实现了 16 位 Unicode 的系统的兼容性。为代孕夫妇保留的0xD800 0xDFFF区域以前属于“私人使用的符号”领域。

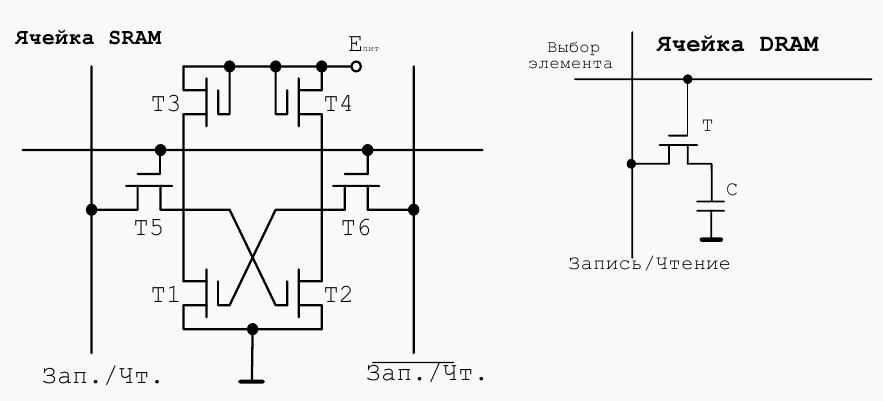
Поскольку в UTF-16 можно отобразить 1 112 064 символов, то это число и было выбрано в качестве новой величины кодового пространства Юникода.

1. 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.

存储单元是计算机存储设备的最小可寻址元件。单元有一个地址（序列号、编号），处理器命令可以访问它们。存储单元由可以处于两种稳定状态之一的元件组成：电容器充电或放电，晶体管处于导电或非导电状态。其中一种物理状态为内存元件产生高电平的输出电压，而另一种则产生低电平的输出电压。第一个通常被视为二进制 1，第二个被视为二进制 0。反向编码也是可能的。尽管从 0 到 1 和从 1 到 0 的转换不会立即发生，但在某些时间点，该信号会达到计算机元素感知为 0 或 1 的值。

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.

处理器寄存器是处理器内的存储器，旨在存储处理器本身运行所需的地址和计算的中间结果或数据。寄存器的特征是一个单数：可以存储在其中的位数。读取存储在寄存器中的信息的作被简化为创建其内容的副本，而原始信息则原封不动地存储在寄存器中。

*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.

总线是将寄存器与另一个寄存器或其他计算机设备连接起来的电路。总线由并联线组成，每根并联线都设计用于传输相应的寄存器。总线还包含几根用于传输同步和控制信号的附加电线。总线仅用于在总线上箭头指示的方向传输信息。特殊方案允许您在某些时候沿总线在一个方向上传输信息，在其他时候在相反方向上传输信息，即组织双向总线。

*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.

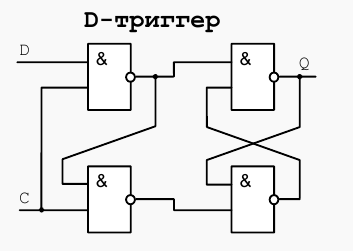
栅极电路是电子钥匙电路，旨在控制从寄存器到总线的信息流，反之亦然。这样的电路包含两个输入和一个输出。信息信号（来自寄存器的数据）被发送到一个输入，控制信号被发送到另一个输入（即门）。如果控制信号为1，则数据通过电路无障碍，如果为0，则没有信息通过电路。多线总线通常用于向栅极电路的输入端提供信息信号。为了传输输出信号，需要具有相同数量电线的总线。如果控制信号为1，则输入和输出总线上的信息信号相同。

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

时钟发生器是一种产生给定频率（通常为矩形）的电脉冲的设备。它用于同步计算机设备之间传输信息的过程。

*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.

功能逻辑电路是一组逻辑元件（根据逻辑代数规则对输入信号执行一种特定逻辑运算的最简单的计算机设备）以及它们之间的连接。

*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.

触发器是一类电子设备，能够长时间保持在两种稳定状态之一，并在外部信号的影响下交替。输出电压值很容易识别每个触发状态。触发器作为功能设备的一个显着特征是记住二进制信息的特性。触发存储器意味着即使在开关信号停止后仍能保持两种状态之一。

*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.

脉冲计数器是一种设备，其输出产生由接收到的脉冲数确定的二进制（二进制十进制）代码。计数器的主要参数是计数模块，即计数器可以计数的单个信号的最大数量。

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

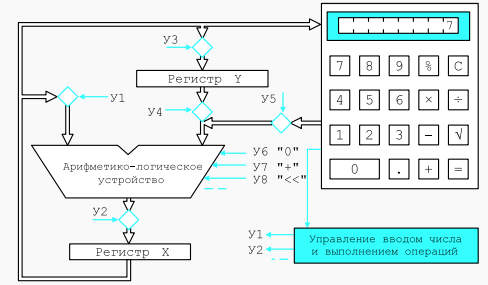
加法器是一种将信息信号（模拟或数字）转换为相当于这些信号总和的信号的设备。

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

计算机功能的结构和原理。以计算器为例的简单处理器的作顺序

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.

典型的计算机由处理器、内存和输入/输出设备组成。  
计算机的“心脏”是处理器，其中包括用于控制从内存中检索指令及其执行的设备、对数据执行作的算术逻辑设备、临时存储处理器数据和状态的寄存器、用于控制和与内存和 I/O 子系统通信的电路。  
输入设备从某些数据载体读取信息，并以其他计算机设备感知的电信号的形式呈现。输出设备以便于视觉感知的形式呈现信息处理的结果。计算机内存包括存储命令和数据的设备。该设备由相同大小的块组成 - 存储单元，旨在存储一个单词的信息。  
存储单元由存储元素组成，每个存储元素的状态对应一个二进制数字。存储在一个单元格元素中的一组零和一代表该存储单元的内容。在微型计算机中，使用未寻址、单播和较少见的双播指令。在单播指令中，从称为累加器的特殊寄存器中选择其中一个作数。作结果也记录在其中。  
未寻址的命令要么指定计算机设备的某些作，要么用于处理具有固定位置的作数（通常使用电池）。在作过程中，计算机依次执行一组相当简单的作：选择命令、确定其类型、执行命令和确定下一个命令的地址。



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 УЗ.

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.

让我们考虑一下最简单的计算机 - 计算器的功能原理。它由两个寄存器组成——X 和 Y，用于存储用户输入和中间计算的结果，一个可以执行最简单的算术和逻辑运算的 ALU，在计算器的功能块之间传输数据的总线和控制阀，控制设备 （CC），键盘和显示器。 显示器不断显示寄存器 X 的内容（跟踪从 X 到显示器的信息总线路径，确保该路径上没有门）。键盘将按下的键的值传输到 U5 阀门，每次按下该键都会启动 UU，根据计算机的当前状态，形成一系列脉冲来执行所需的作，这称为脉冲周期。每组脉冲按顺序发射，时刻与时钟发生器的脉冲重合。 假设用户输入他们想要的号码的第一位数字 （7）。由于这是一个新作，CU 在按下按钮 7 激活后，将输出数字第一位数字的一系列控制脉冲。 首先，需要将 X 寄存器的前一个值保存在 Y 寄存器中。 之后，您需要重置 X 寄存器，为从键盘输入的数字的新数字做好准备。为此，必须关闭所有门，但 Y2 除外 - 它将数据从 ALU 传输到寄存器 Y 和 Y6 - 它们将在 ALU 中形成值 0。 接下来，您需要将值 0 添加到键盘上的数字中。为此，寄存器 X 的内容被发送到 ALU 的右输入 （Y1），将键盘上的数字发送到 ALU 的右输入 （Y5），并选择加法作 （Y7）。 在循环结束时，需要将加法结果传输到 X （U2） 寄存器，并将其显示在显示屏上。

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:

1. 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).
2. Результат записывается в регистр X и показывается на экране калькулятора.

要输入第二个及后续数字，需要在每个输入的数字之后移动 X 寄存器，并将输入的数字与 X 寄存器的移位内容相加。 寄存器 X 的内容通过 Y1 控制的栅极馈送到 ALU 的左侧输入端，而控制寄存器 Y 和 Y5（键盘输入）传输的门 Y4 必须关闭，而 0 则提供给 ALU 的右侧输入端。控制信号 U8 将导致进入 ALU 左侧输入的信息发生偏移。在二进制十进制数系统（计算器通常执行计算）中，小数点后一位的移动对应于 10 的乘以。 偏移结果（控制信号 Y2）写入寄存器 X。 然后您需要将移位结果添加到键盘上的新数字。为此，寄存器 X 的内容被发送到 ALU 的右输入 （Y1），将键盘上的数字发送到 ALU 的右输入 （Y5），并选择加法作 （Y7）。 移位和加法的结果（控制信号 U2）记录在寄存器 X 中。 当用户按下“+”或“=”按钮时，根据此添加是中间作还是最终作，需要添加寄存器 X 和寄存器 Y 的内容。为此： 寄存器X的内容被馈送到ALU的左侧输入（Y1），寄存器Y的内容被馈送到ALU的右侧输入端（Y4），并执行加法作（Y7）。所有其他阀门都关闭（首先，U5 应该关闭）。 结果记录在寄存器 X 中并显示在计算器屏幕上。

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

Unix作系统是作系统内核和文件系统。

*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.

UNIX 是一系列便携式、多任务处理和多用户作系统。UNIX 背后的思想对计算机作系统的发展产生了巨大影响。UNIX 系统现在被认为是历史上最重要的作系统之一。

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.

类 UNIX 系统与其他作系统之间的主要区别在于它们本质上是多用户多任务处理系统。也就是说，同时，许多人可以同时执行许多计算任务（过程）。即使是举世闻名的 Microsoft Windows 系统也不能称为成熟的多用户系统，因为除了某些服务器版本外，一次只能有一个人在一台 Windows 计算机上工作。许多人可以同时在 Unix 上工作，并且他们每个人都可以执行许多不同的计算过程，这些过程将使用该特定计算机的资源。

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

Unix 的第二个巨大优点是它的多平台。系统的核心编写方式使其可以轻松适应几乎任何微处理器

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.

UNIX 还有其他特征：

使用纯文本文件配置和管理系统;

广泛使用从命令行启动的实用程序;

通过虚拟设备（终端）与用户交互;

物理和虚拟设备的文件表示以及一些进程间通信工具。

使用来自多个程序的管道，每个程序执行一个任务。

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.

在 UNIX作系统中，文件是存储为字节流的二进制和符号数据的存储库。

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.

文件的概念是 UNIX 最重要的概念之一。所有可以被用户作的文件都位于一个文件系统中，该文件系统是一棵树，中间顶点对应于目录，叶子对应于文件和空目录。作为文件系统根目录（根目录）的路径为 /。每个目录包含两个特殊的引用文件，一个“.”文件 - 指向当前目录的链接，以及一个指向父目录的“..”链接。

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».

inode - 索引节点 - 文件句柄，其唯一编号。它包含有关文件的所有信息，但文件名和实际文件数据除外。inod 包含：  
文件类型、权限、文件修改/创建时间以及通用名称“元数据”下的其他服务信息

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

Unix作系统包括解释器、标准输入和输出流以及过滤器。

**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).

命令解释器 – 为用户提供与命令行通信的界面的程序;该程序将用户输入的命令“翻译”成作系统可以理解的语言。解释器通常称为 shell。最常见的 shell 是 sh、bash（Unix 中的标准）和 c shell。用户可以单独输入命令，也可以使用一组命令（脚本）输入命令。命令可以直接在命令行上指定，也可以来自标准输入或指定文件。可以将对系统或应用程序实用程序或控制构造的调用作为命令接受。此外，shell 负责重定向 I/O 流。它与一组实用程序一起，是一种作环境、一种编程语言和工具，用于解决系统和应用程序问题，特别是在自动化执行的指令序列方面。  
要与用户交互和交换信息，您可以使用称为标准输入（用于从中读取）和输出（用于写入）流的文件。屏幕上的输出也表示为写入文件，输入也表示为从文件读取。除了输入和输出流之外，还有一个标准错误流，它输出所有不应进入输出流的服务信息（错误消息或程序进度）。  
Стандартные потоки привязаны к *файловым дескрипторам* с номерами:  
0 для ввода (stdin) 1 для вывода (stdout) 2 для ошибок (stderr).

标准流绑定到带有数字的文件描述符：  
0 表示输入 （stdin），1 表示输出 （stdout），2 表示错误 （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:

默认情况下，线程与终端（命令行）相关联，但它们可以连接到任何内容——文件、程序或设备。在解释器中，此作称为重定向。因此，标准流不仅可以重定向到文件，还可以重定向到其他程序的输入。  
以下作用于执行重定向：

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.*

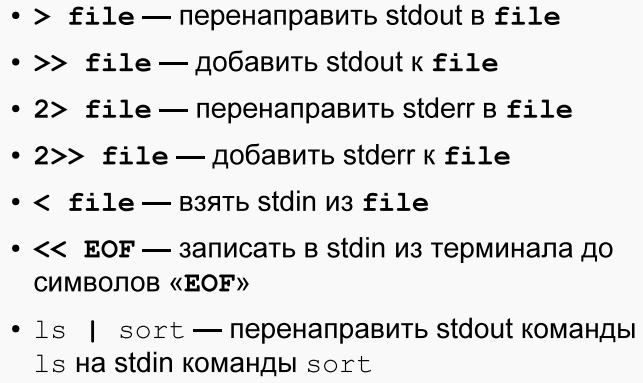
*命令>文件（或>>）  
执行命令，并将输出放入文件中（或添加到末尾）。*

*命令 < 文件 文件用作输入源。在这种情况下，程序的每个输入请求都会从文件中读取 1 行文本。*

*命令 1 | 命令 2 命令 1 的输出将作为命令 2 的输入，而不使用中间文件。此功能称为管道。*

*命令 2> 文件  
错误流定向到一个文件。默认情况下，此流输出为标准输出。*

*命令 2>&1 文件（或 &> 或 >&）  
此语法用于组合输出流和错误流以将它们一起处理。*

**

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».

所谓的“空设备”文件 - /dev/null - 重定向到它可以让您摆脱不必要的错误消息或忽略输出。您还可以使用它来创建空白文件作为输入源。当写入其中时，它可以容纳任意数量的信息，它就像一个“黑洞”。

Фильтры:

过滤器：

wc, grep, sort

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

Unix作系统 - 基本命令、文件权限以及如何设置它们。

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 с 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 катеория.*

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:

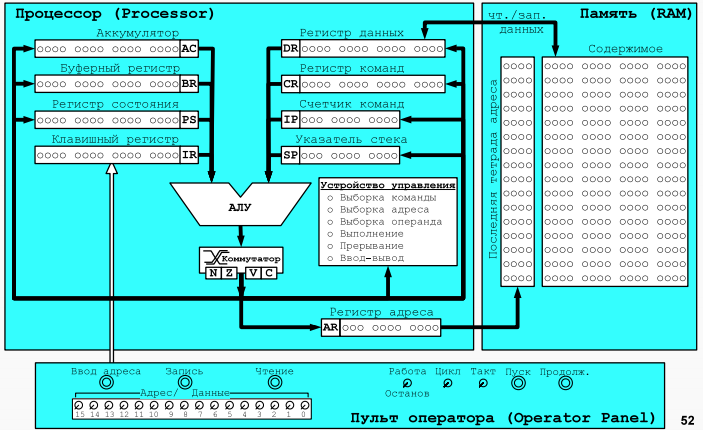
* **u+x** - разрешить выполнение для владельца;
* **ugo+x** - разрешить выполнение для всех;
* **ug+w** - разрешить запись для владельца и группы;
* **o-x** - запретить выполнение для остальных пользователей;
* **ugo+rwx** - разрешить все для всех;

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.

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

BEVM 的组成和结构。BEVM 的地址空间。BEVM的指令系统，指令格式。机器循环。



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.

*控制设备以微程序控制单元 （MCU） 的形式设计 - 一台简单的计算机，其程序直接由微作组成 - 即计算机门值的逐时钟变化，设置原子作：ALU 中的计算、寄存器之间的数据传输和最简单的检查。LMOU 的程序代码称为微码。*

*LPM 执行 BEVM 的所有机器命令。在 LUT 中执行机器指令称为指令循环。命令周期在逻辑上分为五个周期：*

*团队抽样周期。将可执行指令加载到指令寄存器中并对其进行部分解码。它们针对每条可执行指令执行。*

*地址采样循环。它旨在处理地址命令并获取作数的地址，同时考虑寻址模式。 作数采样周期。对于那些团队，如有必要，将指令的第二个作数放在 BoJ 中。我们记得，第一个是电池。*

*执行周期。命令被执行。*

*中断循环。如果允许中断并且 I/O 设备已准备好交换（即需要中断，这将在后面讨论），则执行循环。*

*为确保 BEV 操作员的工作，它提供了控制台作周期的微程序实现：*

*地址输入 – 将密钥寄存器中的地址输入到命令计数器中。*

*读取 — 从 BL 中的内存中读取 1P 中地址的信息，1P 增加 1。*

*记录 — 来自 1P 的信息记录在 EP 地址的内存中，1P 增加 1。用于操作员模式下的程序和数据输入。*

*启动后重置 BEV 状态并继续执行程序。*

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.

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

计算机系统中的计算组织。班次、算术和逻辑运算。团队抽样周期。

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.

无符号二进制整数可用于表示零整数和正整数。在 16 位字中，它们可以在 0 到 65535 之间变化。这些是定点数。当需要区分正数和负数时，使用有符号二进制整数。负数以附加代码表示。这简化了计算机的设计。使用 ADD 命令在基础计算机中执行带符号和不带符号的二进制整数的添加。如果发生从最高有效数字 A 的传输，则将 1 输入到传输寄存器中，否则将 0 输入其中。减法可以通过添加可递减代码和减法的附加代码来完成。在基本计算机中，没有执行乘除的指令（ALU 不执行此类运算），因此必须通过编程方式获取乘积和商。要更改数字的符号，您需要将其反转，然后在最低有效位 （NEG） 上加一个。按位数据处理由逻辑乘法指令、循环移位以及传输寄存器反转和刷新指令提供。

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.

AND 命令对累加器每位和单元内容执行 AND 运算。仅当两位都等于 1 时，对每对作数位执行命令的结果才为 1，而在其他情况下，结果位为 0，即该命令允许您选择或清除单词的某些位。

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.

ROL 和 ROR 命令关闭电池和环传输寄存器，并将环的所有位向左或向右移动。数字的算术移位（ASL 和 ASR）可用于实现乘法或除法运算 2（一班）、4（两班）、8（三班）等。

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

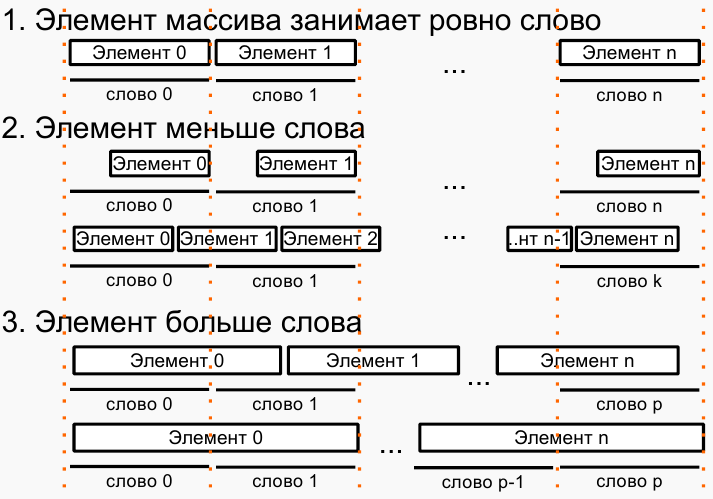
if CR(12) = 0 then GOTO ADDRLESS @ 78

GOTO IO @ C2

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

数据数组的组织。寻址模式。获取 BEVM 的地址和作数的循环。





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

控制 BEVM 中的计算过程。分支命令，LOOP 命令的循环。

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).

控制计算过程的问题，即所需的指令执行顺序，在基本计算机中借助转换命令、“增量和跳过”（LOOP） 和“停止”（HLT） 命令得到解决。

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



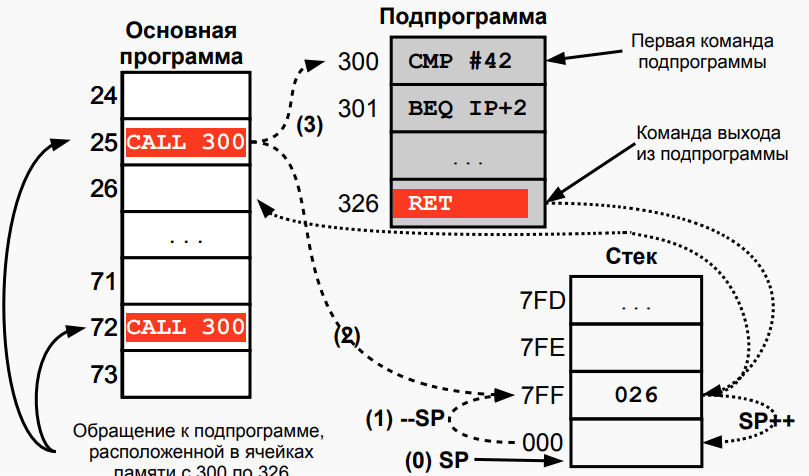
1. Подпрограммы в БЭВМ. Цикл исполнения команд перехода и возврата из подпрограммы. Стек, передача параметров. Позиционно-независимый код. Загрузчик и библиотеки.

BEVM 中的子例程。从子例程执行跳转和返回命令的周期。堆栈，参数传输。与位置无关的代码。加载器和库。

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.

程序的各个部分执行相同的数据处理作的情况并不少见。在这种情况下，程序的重复部分被分离成一个子程序。在基本计算机中，CALL 和 RET 命令用于此目的。

<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

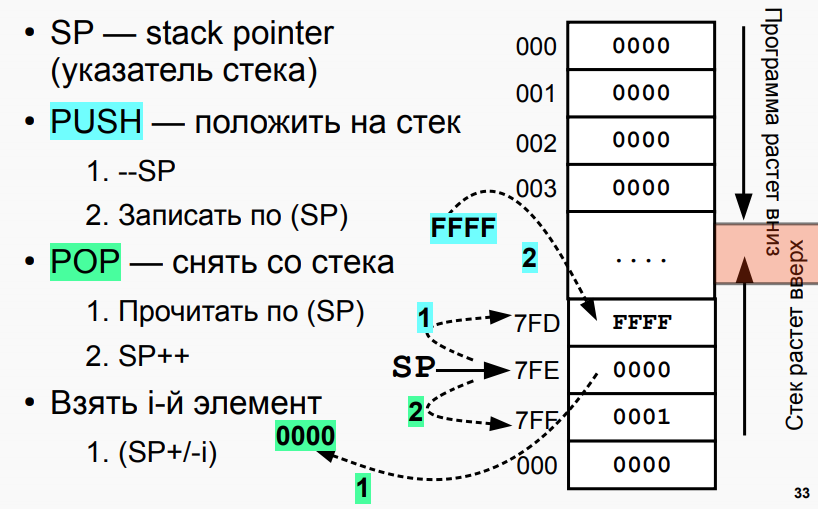
● SP → AR; Вершину стека поместить в AR

● MEM(AR) → DR; прочитать адрес возврата

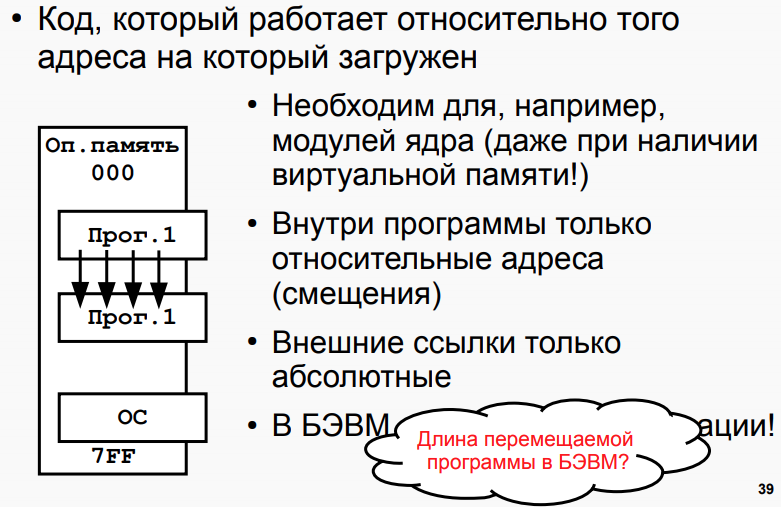
● DR → IP; вернуться из подпрограммы

● 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

下载器和动态程序链接器： 任何作系统都有相应的程序或内核的一部分

– 启动到您选择的地址（即使在虚拟内存中）

– 更改程序中地址的常量部分

– 加载基本寄存器值

– 动态加载共享库

– 将主程序的地址链接到被调用的库

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

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

计算系统中 I/O 的组织。发起交换、传输信息并完成交换。司机。

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).

可以将大量各种输入/输出设备或外部设备 （VU） 连接到计算机。这些设备向计算机传输并从计算机接收大量信息，这些信息不能仅放置在处理器的寄存器中。因此，信息从 VU 传输到计算机内存，并从其内存传到 VU。在这种情况下，交换可以在计算机程序的控制下通过处理器的寄存器进行（软件控制的数据传输）或在特殊外部设备（直接内存访问控制器）的控制下进行，绕过处理器（直接访问内存的数据传输）。

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.

软件控制的交换以小部分进行，直接访问内存，信息以大块形式传输。使用软件控制的交换时，必须创建一个程序，确保将数据从计算机内存传输到累加器，然后传输到 VU 控制器的内存寄存器（数据输出）或从 VU 控制器的数据寄存器传输到累加器，然后传输到计算机内存（数据输入）。在此程序中，您可以实现三种类型的交换之一：同步、异步和中断。

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

数据传输：同步/异步

完成交换并由驱动程序（程序）接收结果：同步/异步

驱动器：

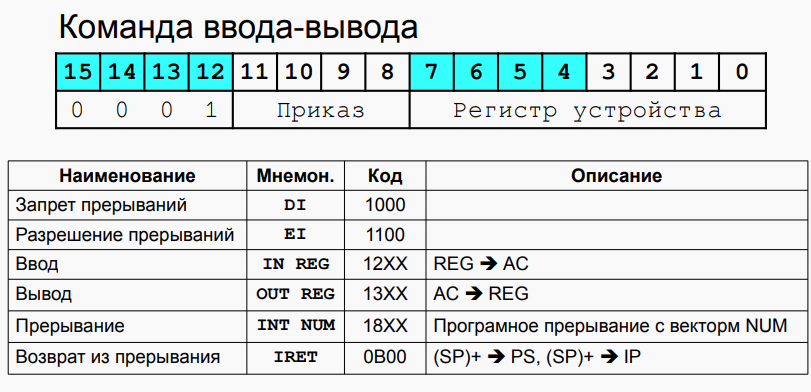
● 与设备协作

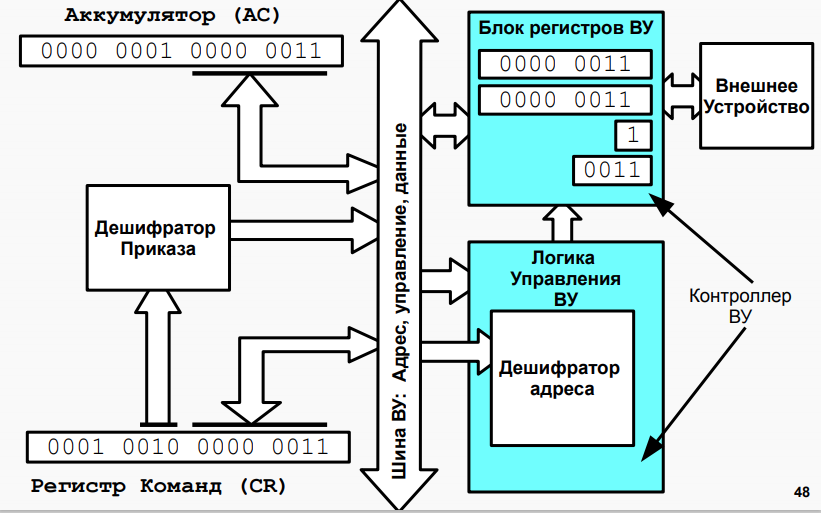
● “了解”设备的工作原理、寄存器地址、支持的工作模式

● 由统一的软件界面控制

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

BEVM 中的 I/O 组织。I/O 设备、命令。





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

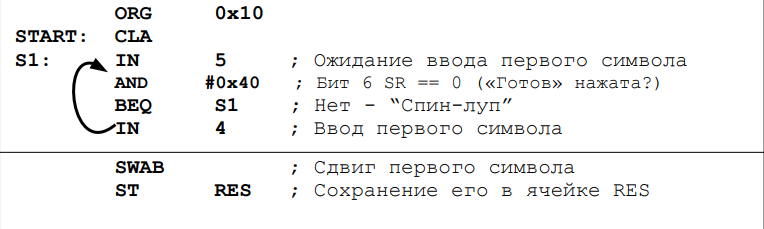
最简单的控制器安装在 VU 和处理器之间，每个控制器都包含：数据寄存器（用于 VU 和处理器之间的数据交换）、地址解密器（允许您在连接到处理器的 I/O 设备的所有请求中区分对此 VU 的访问）、控制逻辑（处理器执行某些作的解码顺序）和状态寄存器（存储有关 VU 是否准备好与处理器交换数据的信息）。最简单的 VU 控制器通常使用单位状态寄存器，通常称为标志。VU 控制器通过输入和输出总线、地址总线和控制总线连接到处理器，这些总线传输来自处理器的命令和有关 VU 状态的信息。 VU 设备：输出、输入、I/O、I/O PRO、定时器、打印机、爬行线、7 次指示灯、键盘、数字键盘

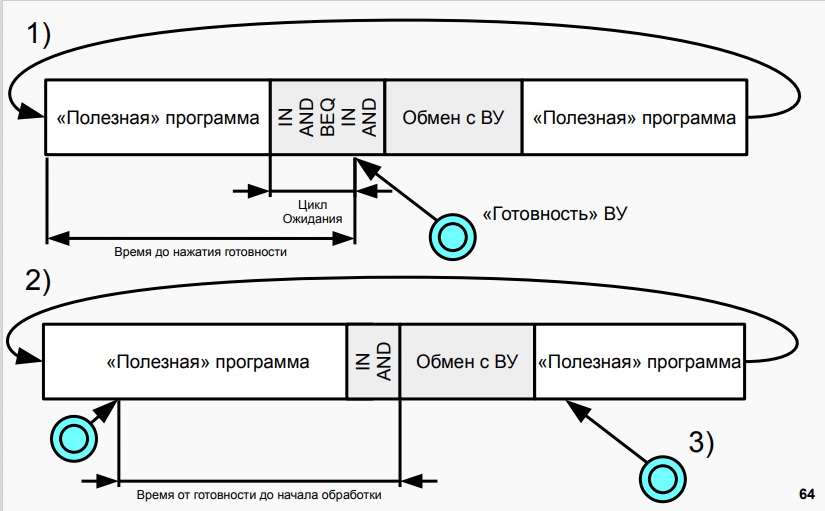
1. Организация асинхронного обмена в БЭВМ. Пример программы. Временные издержки асинхронного обмена.

在 CEVM 中组织异步交换。程序示例。异步交换的时间开销。

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.

异步交换程序的算法：首先，检查VU的交换准备情况，如果准备就绪，则发出交换命令（输入或输出）。VU 通过在 VU 控制器中设置标志来报告就绪情况。在异步交换中，计算机必须花时间等待准备就绪的时刻，并且由于准备就绪是通过命令检查的，因此计算机此时无法执行任何其他数据转换工作。





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

组织 BEVM 中的中断。中断向量，中断控制器。

**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.

中断是来自软件或硬件的信号，用于通知处理器需要立即注意的事件的发生。在 BEVM 的背景下，这是准备与某些 VU 交换数据的信号。

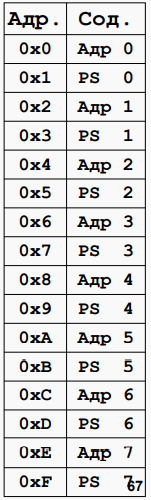
Также стоит отметить преимущества над асинхронным режимом передачи данных (если Вы помните, то реализовывать его можно двумя способами: через spin-loop или одну проверку с последующим окном ввода-вывода в циклической полезной программе):

还值得注意的是与异步数据传输模式相比的优势（如果您还记得的话，它可以通过两种方式实现：通过自旋循环或单次检查，然后是循环实用程序中的 I/O 窗口）：

* 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
* 处理器不会闲置等待 VU 准备就绪
* 同时与我们需要的所有 VU 一起组织工作
* 如果所有 VU 都没有准备好交换，那么处理器正忙于有用的工作

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.

中断的一个显着特点是必须在中断时保存处理器状态，这样在处理中断时，我们可以返回主程序并继续其执行，而不会发生各种冲突。



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

中断向量：

● 中断处理程序地址聚合和状态寄存器（PS）

● 必须先初始化才能开始中断处理 “

– 至少将它安装在一个不执行任何作的子程序上

– 作系统和 BIOS 的责任

● 在 BEVM-NG 细胞中，0x000 0x10

– 总共有 8 个向量，每个向量两个单词

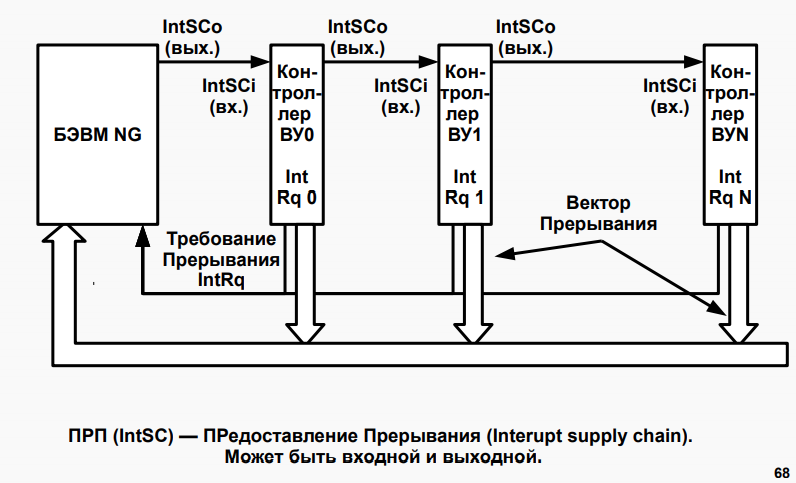
– 同一矢量上可以有多个中断

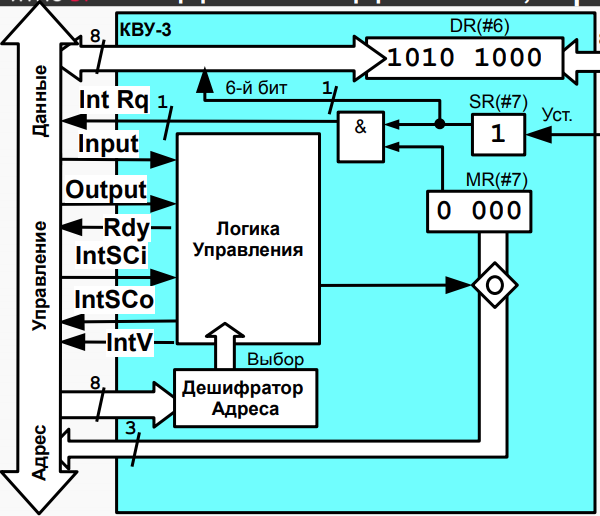
Регистр управления или Management Register — регистр, хранящий в себе информации о разрешении/запрете прерывания от данного ВУ и вектора прерывания, к которому привязано ВУ.(MR)

管理寄存器是一个寄存器，用于存储有关允许/禁用来自给定 VU 的中断以及 VU 绑定到的中断向量的信息。

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

3 个最低阶位包含中断向量号，如果允许来自此 VU 的中断，则第 3 位将包含 1，如果禁止中断，则第 3 位将包含 0。





* 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.

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

组织 BEVM 中程序中断的交换。程序示例。中断循环。

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.

BEVM 中有 8 个中断向量可用，它们位于存储单元中0x0 0xF（含）。每个中断向量可以有多个中断。在开始之前，必须将中断向量加载到 VU 中并允许中断。

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)

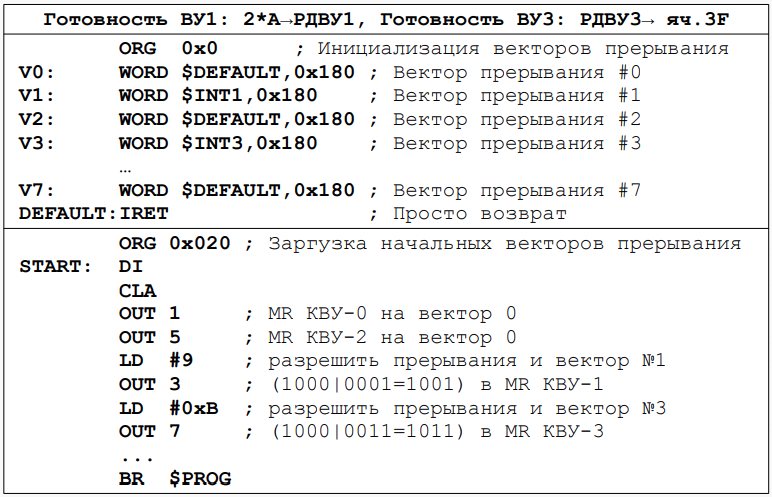
11) MEM(AR) -> DR; в DR адрес обработчика прерывания

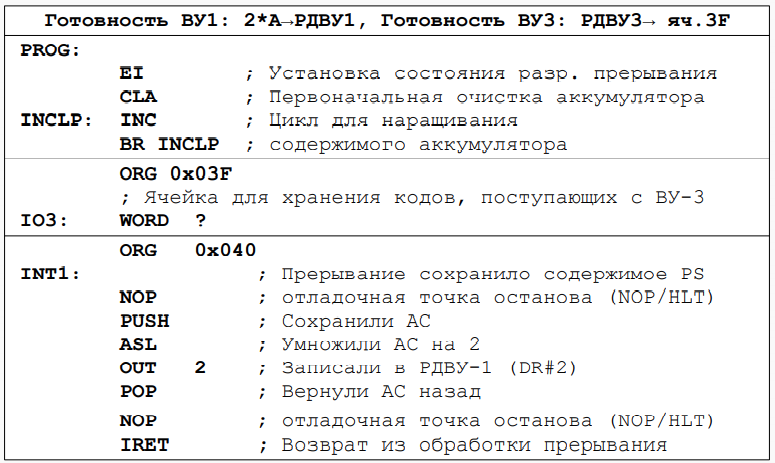
12) DR -> IP

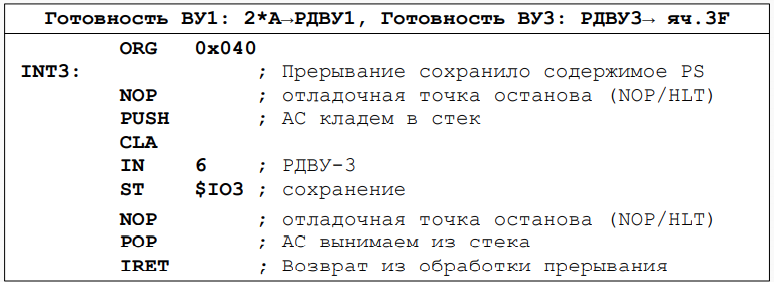
13) LTOL(BR + 1) -> AR ; в AR адрес, в котором лежит PS для обработчика прерывания

14) MEM(AR) -> DR

15) DR -> PS







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

多级计算机的概念。不同级别的程序的概念和示例。

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.

在计算机上执行用算法语言编写的程序的能力是在特殊系统程序的帮助下提供的：编译器和解释器。编译在于执行算法的过程只有在翻译源程序的过程完成后才进行。然而，在解释中，原始程序的每个运算符在执行前都会被解释器程序替换，并立即使用等效的机器指令序列。与编译不同，在解题的同时进行解释时，机器需要原始程序和解释程序。

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.

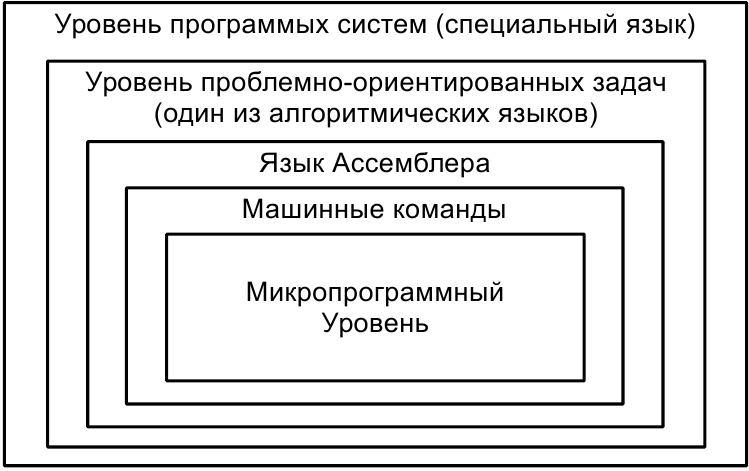
创建编译器（解释器）的成本和翻译程序过程所花费的时间很大程度上取决于编译语言和生成语言的相似性。因此，算法语言不会立即将程序翻译成机器指令的语言。编程语言有一定的层次结构，其中更复杂的语言基于前一种语言。

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.

中间语言的一个例子是符号指令编码语言，通常称为汇编语言。汇编语言（为每种类型的计算机开发）是计算机技术中自动化编程的第一种方法。它们允许使用象征性名称和标签。来自此类语言的编译器称为汇编器。它们为符号变量分配某些存储单元，组织程序不同部分之间的连接，与指令级编程相比，这极大地方便了编程。

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.

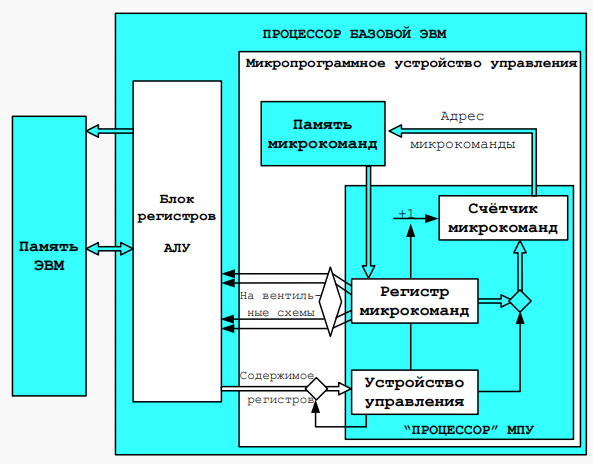
使用特定语言使用计算机的人通常认为他与计算机通信的语言是它的机器语言。因此，对于同一台计算机的不同用户来说，他们似乎在不同的计算机上工作。因此，虚拟（看似）计算机和多级计算机的概念。

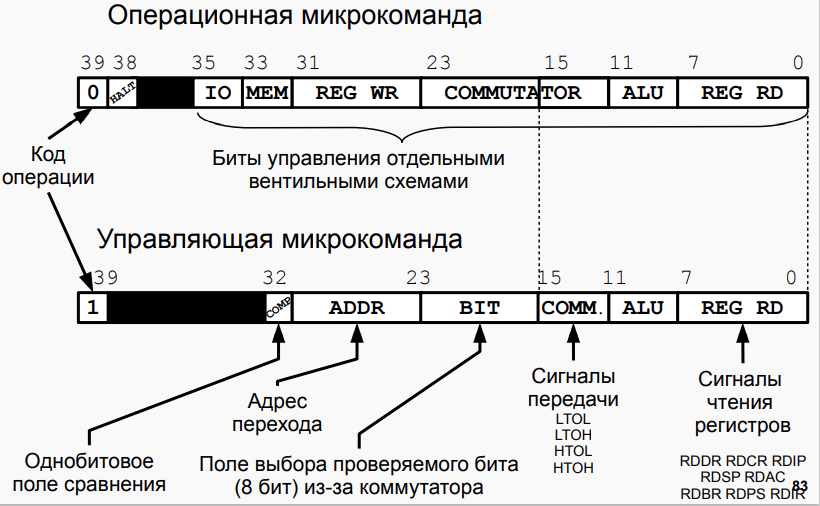
**Многоуровневая ЭВМ** – это вычислительная машина, имеющая средства для работы с n различными уровнями языков программирования. Нижний язык, или уровень, является наиболее простым, верхний –наиболее сложным. Такую машину можно рассматривать как n различных виртуальных машин, каждая из которых имеет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

多级计算机是能够处理 n 种不同级别的编程语言的计算机。较低的语言或级别是最简单的，较高的语言是最困难的。这样的机器可以被认为是 n 个不同的虚拟机，每个虚拟机都有自己的机器语言。这些虚拟机的硬件实现的复杂性随着层数的增加而增加。

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

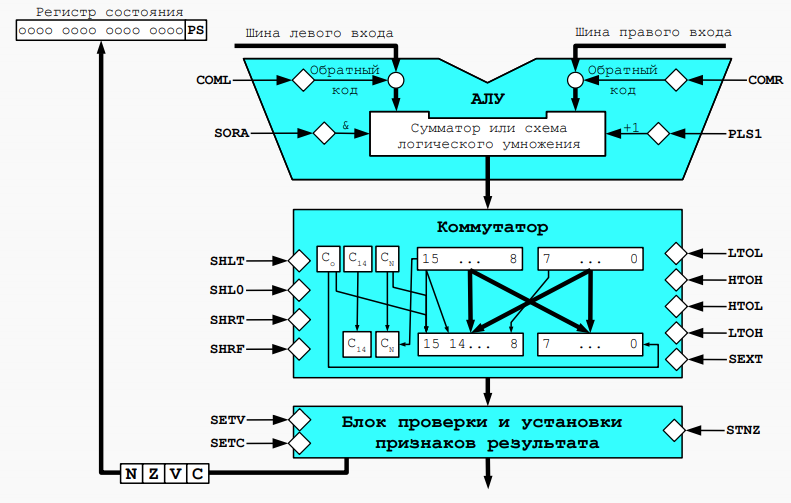
BEVM 的固件级别。微型程序的结构。微指令格式。

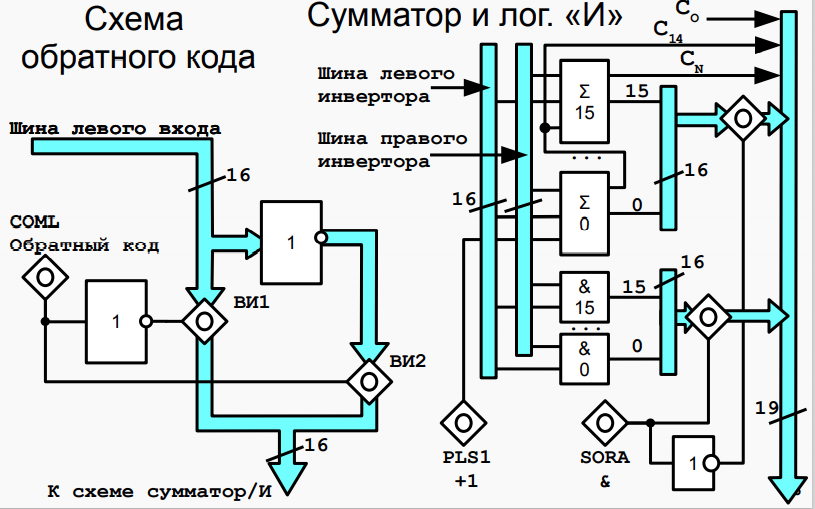


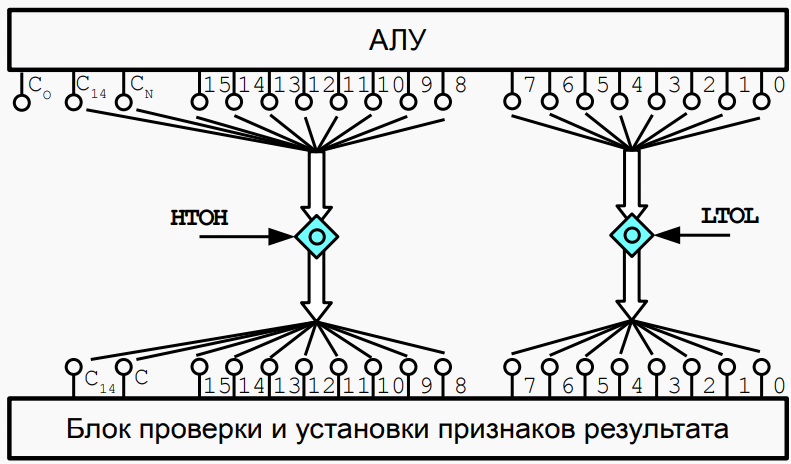


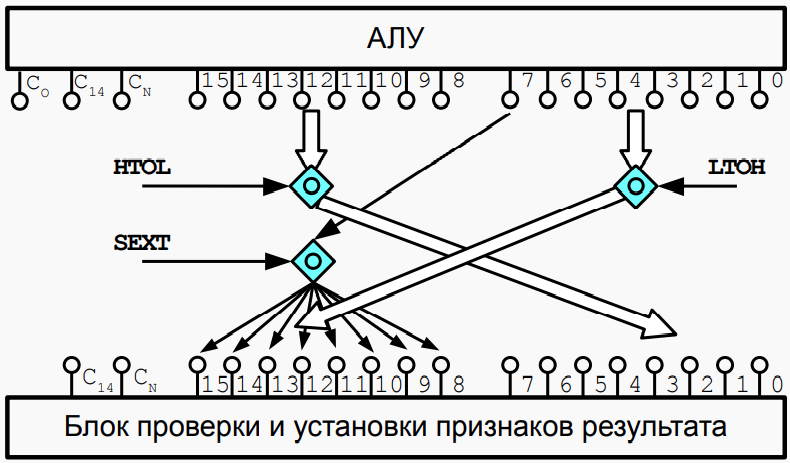
1. Структура и принципы работы арифметико-логического устройства и коммутатора. Регистр состояния БЭВМ

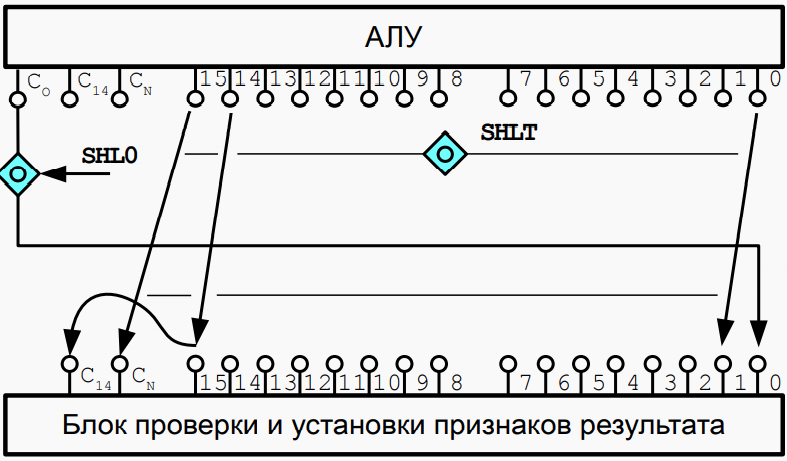
算术逻辑器件和换向器的结构和工作原理。计算机状态寄存器

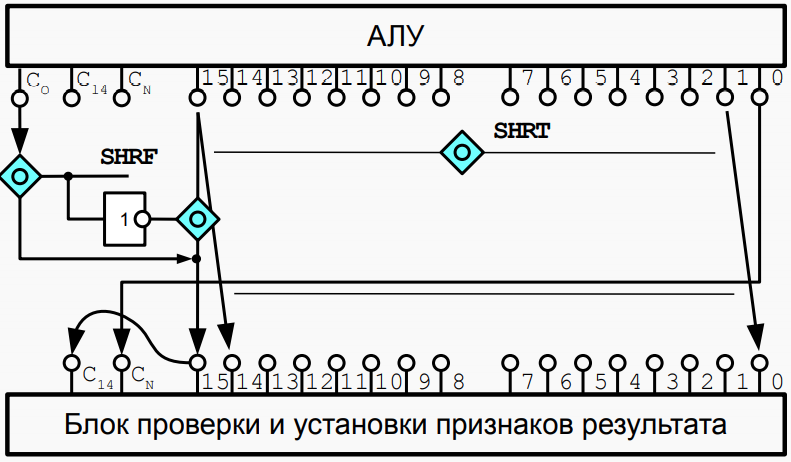


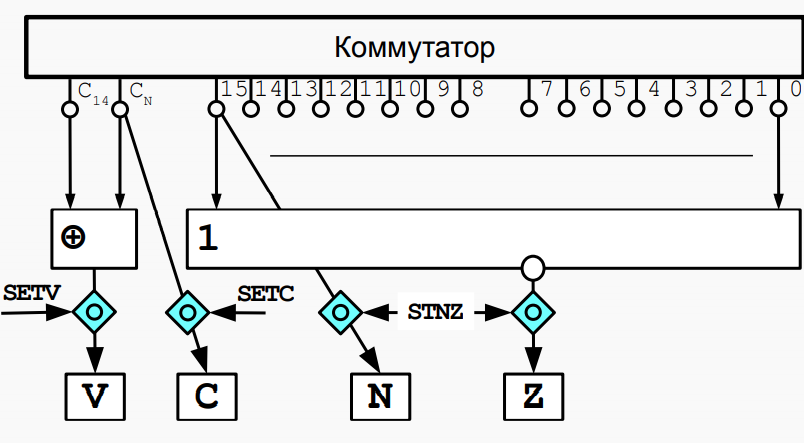


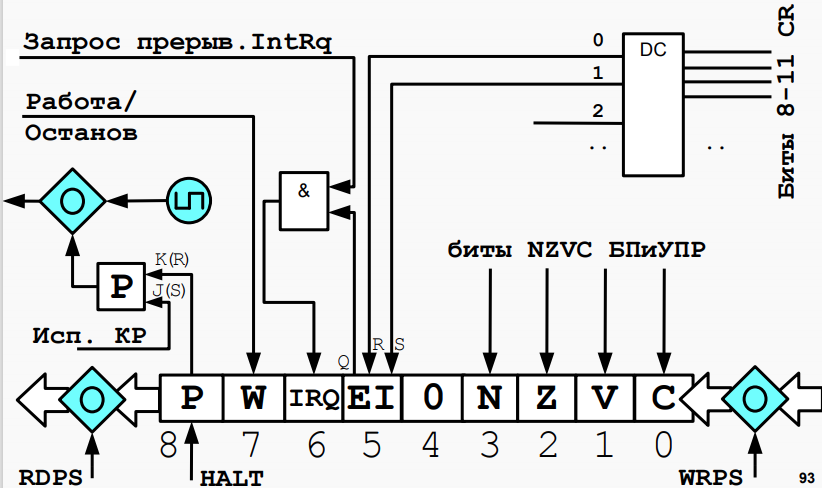




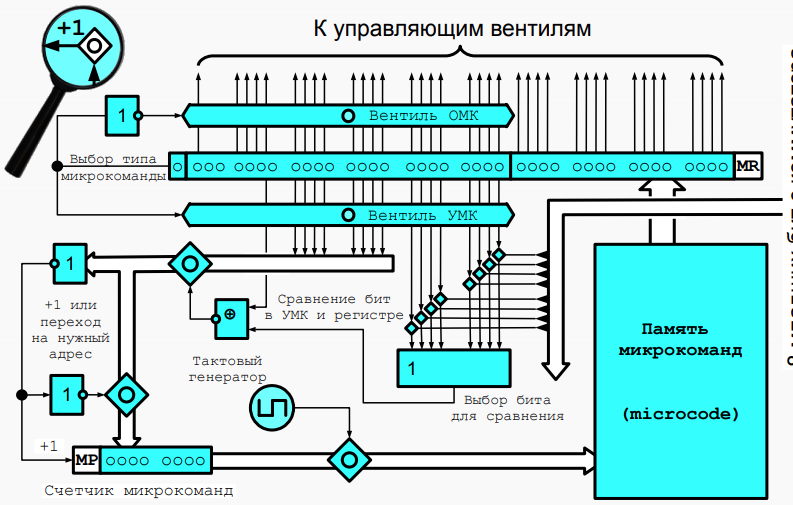








1. Микропрограммное управление вентильными схемами. Схема управления. Интерпретатор БЭВМ.



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

BEVM 解释器

● 256 个微指令存储插槽，包括冗余

● 包含水平微命令

● 命令采样周期

●作数地址采样环路和寻址模式处理

●作子采样环路

● 执行周期

– 地址命令的解码和执行

– 解码和分支

– 解码和执行未寻址的指令

● 解码和 I/O 命令执行

● 中断循环

● 远程操作

● 自由单元：

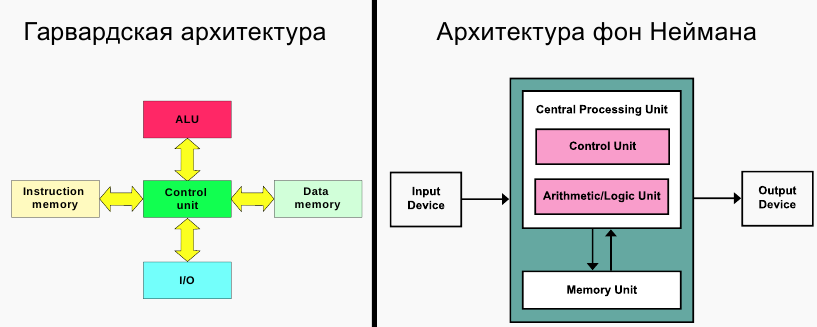
– 算术命令

– 跳跃命令

– 无地址的命令

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

计算机架构。哈佛和冯·诺依曼的建筑。使用总线组织计算机架构的交换。



● Первое поколение — электронные лампы

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

● Второе поколение — транзисторы

– 5Э92б, 1964, самодиагностика, горячая замена, 500000 оп/c – 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 РDP-11

● Четвертое поколение — сверхбольшие интегральные схемы

– Эльбрус — разработка по настоящее время

● 第一代 – 真空管

– 列别杰夫，1950 年，MESM – BESM，1953 年，BESM – 10000 op/s，53 kW。

● 第二代 - 晶体管

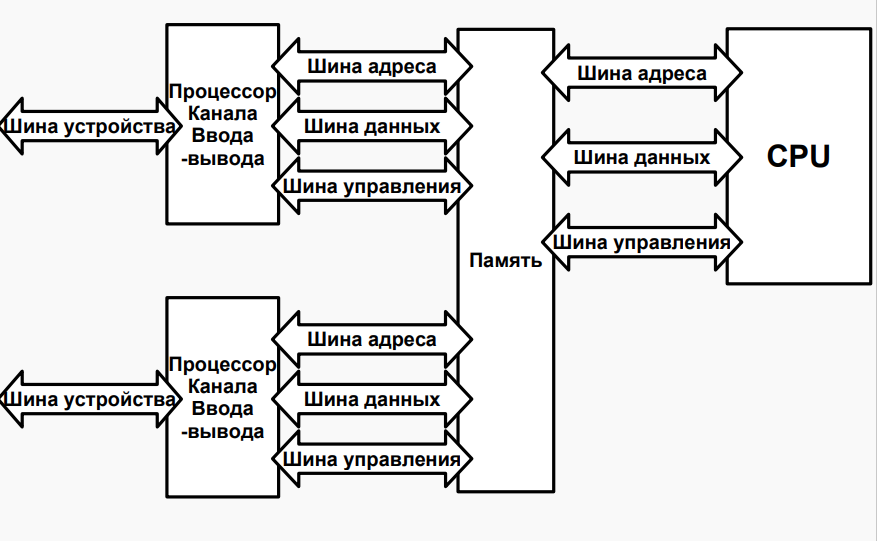
– 5E92b，1964 年，自诊断，热插拔，500000 运算/秒 – BESM-6,1965 年，+ 共顶加工，电话远程控制。线

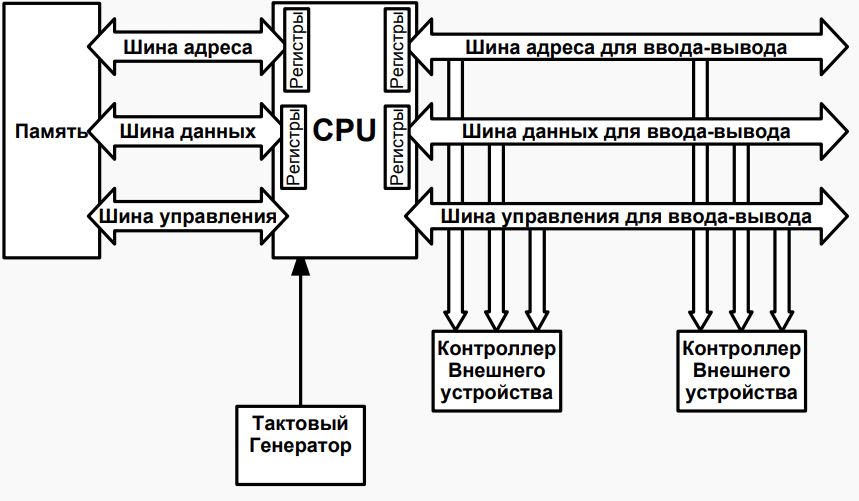
● 第三代

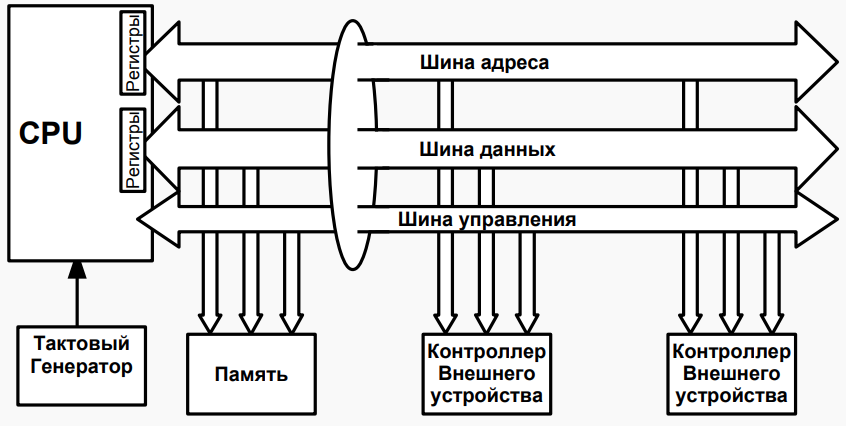
—集成电路 – Ryad 指令，1968 年，S/360 克隆，1971 年 – ES 计算机 – PDP-11 克隆

● 第四代 – 超大型集成电路

– 厄尔布鲁士 – 迄今为止的发展



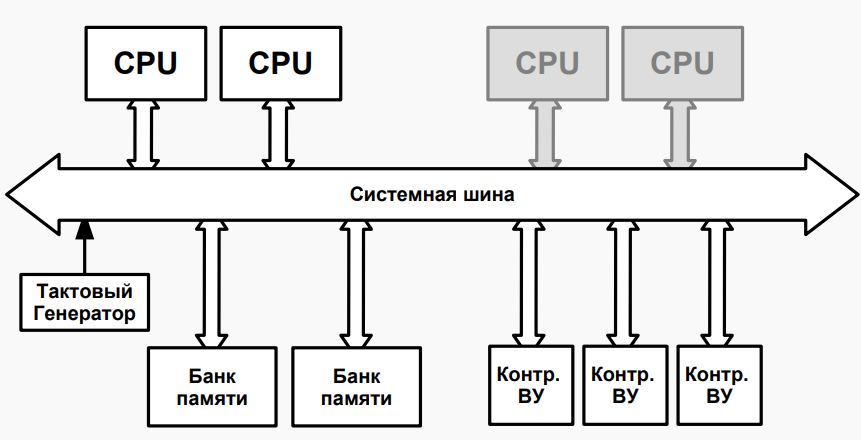


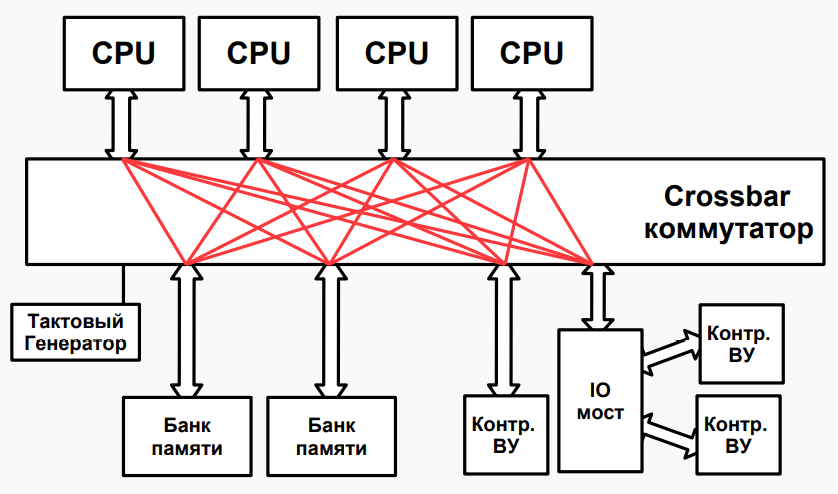


1. Архитектура многопроцессорных ЭВМ. Системный коммутатор. Архитектуры UMA и NUMA.

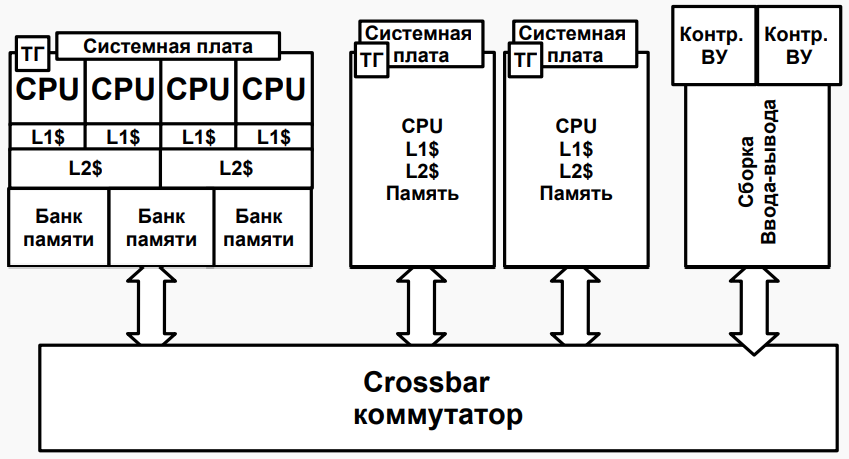
多处理器计算机的架构。系统总机。UMA 和 NUMA 架构。

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.

UMA（统一内存访问）是用于多处理器系统的共享内存架构。该模型使用单个内存，该内存由所表示的多处理器系统中的所有处理器使用互连网络访问。每个处理器具有相同的内存访问时间（延迟）和访问速度。它可以使用单总线、多总线或交换机。

**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.

NUMA（非统一内存访问）也是一种多处理器模型，其中每个处理器都与分配的内存相关联。但是，这些小块内存被组合成一个地址空间。要考虑的主要事情是，与 UMA 不同，内存访问时间取决于处理器所在的距离，这意味着内存访问时间会发生变化。这允许您使用物理地址访问任何内存位置。

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

现代处理器的结构。处理器环境。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

● 16/32/64位位地址和数据宽度

● 时钟速度为500MHz-5GHz。

● 多处理器 1-100+ CPU

● 多核1-16核

● 1GB 到 TB 的 RAM

● 使用不同级别的缓存

● 超标量

● CISC、RISC、VLIW

● Complex Instruction Set Computer (CISC)

– 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.

● 复杂指令集计算机（CISC）

– 受兼容性影响的传统处理器（例如英特尔）

处理器中存在相对较少的通用寄存器;

大量的机器指令，其中一些是由复杂的核操作员实现的;

处理作数的多种方法; 各种位大小的多种指令格式;

存在处理与内存访问相结合的命令。

● Reduced Instruction Set Computer (RISC)

– 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.

精简指令集计算机 （RISC）

– 简单的指令集，每个报价执行指令 这个想法是将 VM 指令列表限制为最常用的简单指令，这些指令仅对位于处理器寄存器中的数据进行作。只有在特殊读写命令的帮助下才允许内存访问。用于指定作数地址的指令格式和方法的数量已大大减少。瞄准速度。

● Very Long Instructions Word (VLIW)

– Несколько инструкций, упакованных в одну команду

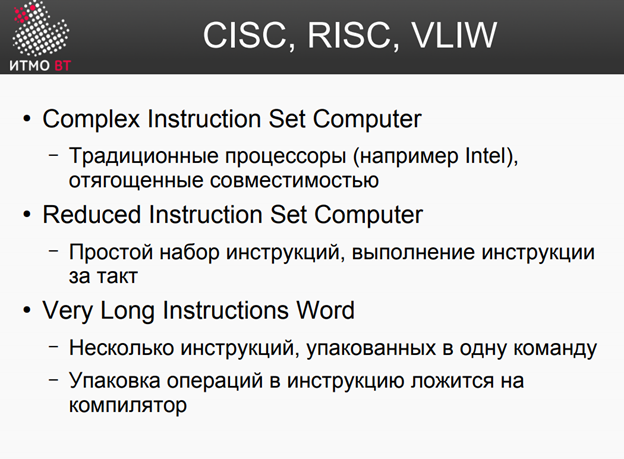
– Упаковка операций в инструкцию ложится на компилятор

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.

超长指令字 （VLIW）

– 将多个指令打包到一个团队中

– 编译器将作包装到指令中 VLIW概念基于RISC架构，将几条简单的RISC指令组合成一条超长指令并并行执行。  
VLIW 背后的想法是，由一个“合理”的编译器来有效地调度并行执行多个指令。这样的编译器首先检查原始程序，以查找可以同时执行的所有命令，并且以不会引起冲突的方式。在分析过程中，编译器甚至可以部分模仿相关程序的执行。在下一步中，编译器尝试将此类命令组合成包，每个包都被视为单个超长指令。



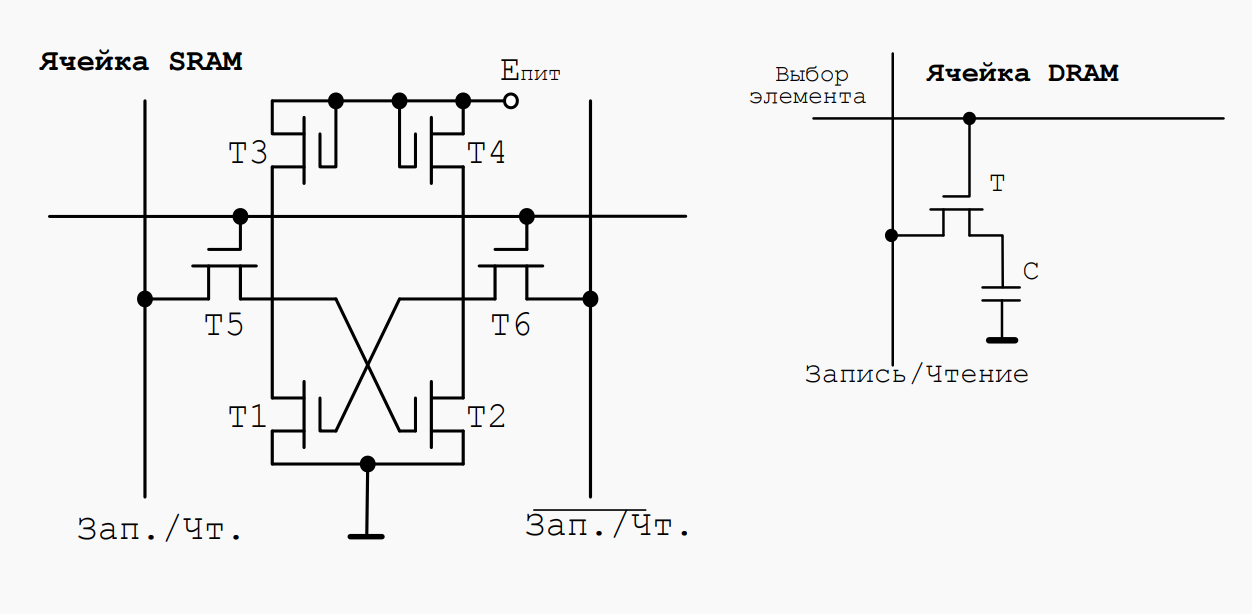
**Структура современного процессора**

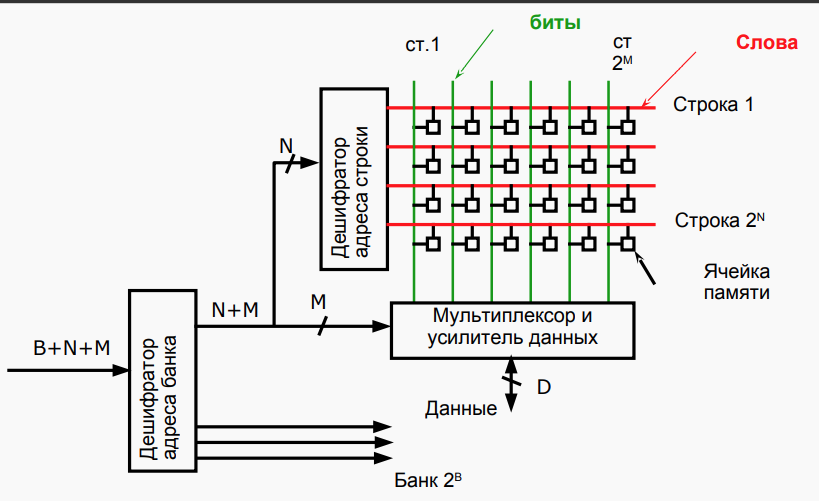
Ядро процессора – это его основная часть, содержащая все функциональные блоки и осуществляющая выполнение всех логических и арифметических операций.

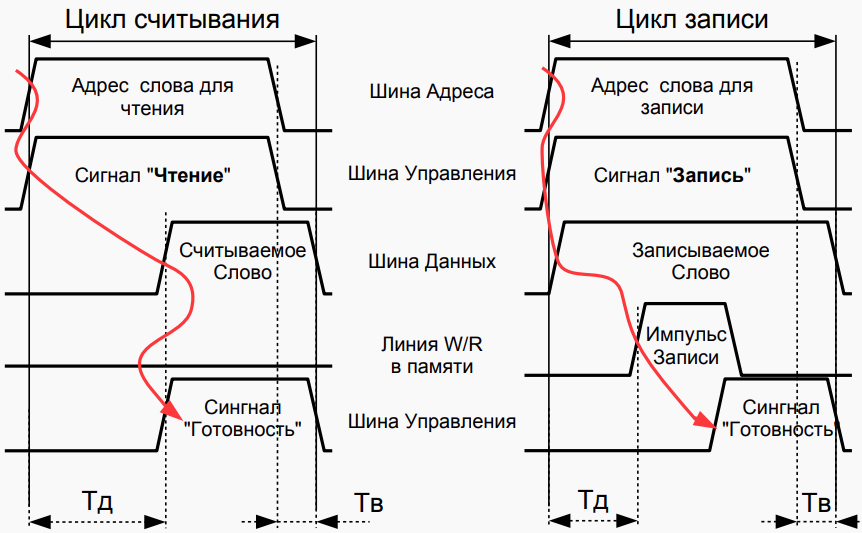
现代处理器的结构 处理器的核心是它的主要部分，它包含所有功能块并执行所有逻辑和算术运算。

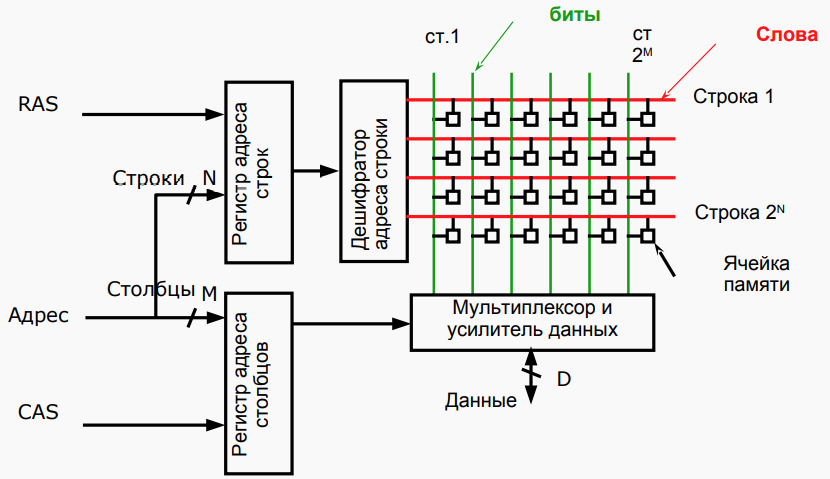
1. Адресуемая память, организация и временные диаграммы. Конструктивные особенности современной памяти.

可寻址的内存、组织和时间图。现代记忆的设计特点。









Конструктивные особенности современной памяти:

● Burst mode — пакетный режим

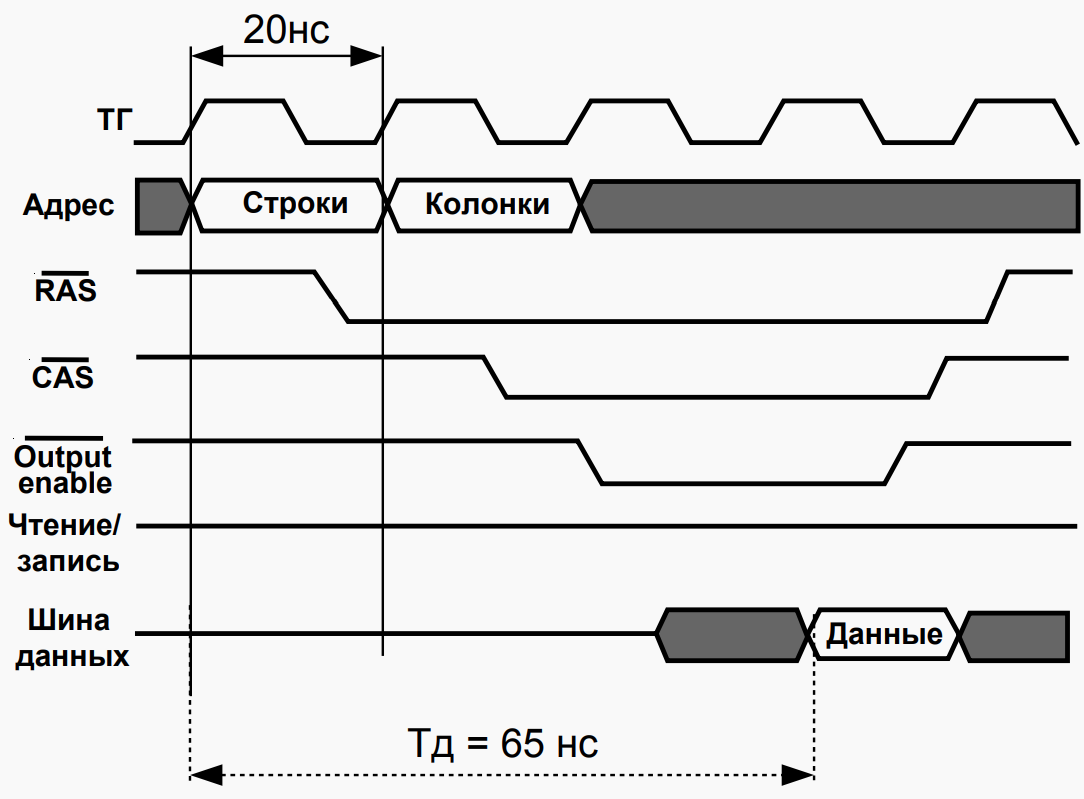
● Double Data Rate — передача данных и по фронту и по спаду

● SPD — чип, содержащий идентификационную информацию

● Interleaving — расслоение памяти, повышает производительность

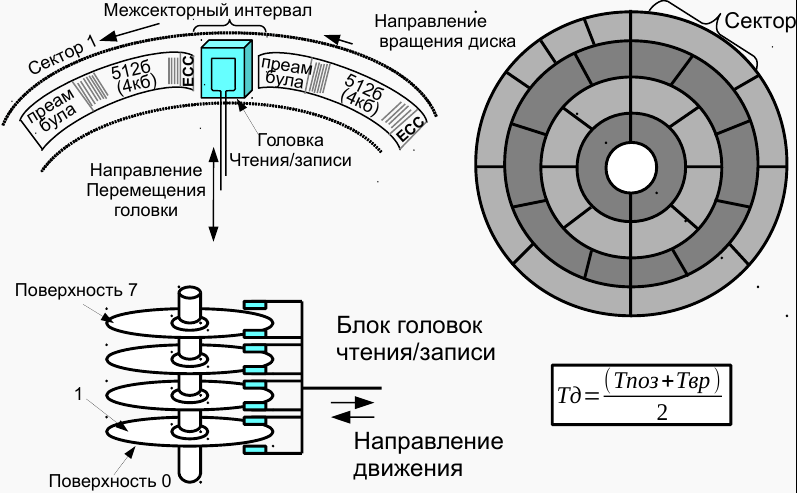
● DDR4-2133 8192MB **PC4-17000** индекс производительности

* 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.
* 现代记忆的设计特点：
* ● 突发模式
* ● 双倍数据速率 — 沿正面和下坡进行数据传输
* ● SPD – 包含识别信息的芯片
* ● 交错 – 内存分层，提高性能
* ● DDR4-2133 8192MB PC4-17000 性能指标 在动态存储芯片中包含一些静态存储器; 内存和 CPU 的同步运行，即使用内部管道架构和交错地址。 连拍模式——在里面。从内存中请求数据不是以单个字节的形式，而是以由 32 位或 64 位组成的数据包的形式，即在 1 个周期内读取 2 或 4 个机器字; 交错模式基于逻辑链接字节一个接一个地位于内存中的事实。在内存再生时，处理器将无法使用它。为了避免内存作中的此类暂停，数据被放置在各种内存芯片中。因此，当数据在其中一个芯片中重新生成时，处理器可以同时从另一个芯片读取字节; 寻呼模式 – 基于与前一种模式相同的原理。具有相同行地址的所有单元格都被分组到所谓的页面中。由于行的地址不会更改，因此可以节省搜索和读取数据的时间。通常，内存分为 512 字节或更多的页; 内存缓存用于加快对 RAM 中数据的访问速度。为此，在 CPU 和 RAM 之间放置了一个小型高速缓存，通常在 2 MB 以内，以 CPU 频率运行，这意味着它不需要等待周期。



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

面向写入的内存（块内存）。磁盘存储器和磁带存储器的组织。



1. Характеристики запоминающих устройств. Пирамида памяти.

存储设备的特性。记忆金字塔。

* 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
* 位置 – 处理器、内部、外部
* 电容 – 公制 （kilo-） 和二进制 （kibi-） 因子
* 转发单元 – Word、缓存行、磁盘块
* 访问方法 – 任意（寻址）、面向记录（直接）、顺序、关联

Характеристики памяти

1. Быстродействие и временные соотношения
   1. Время доступа Тд
   2. Длительность цикла памяти (время обращения) Tц
   3. Время чтения и время записи
   4. Время восстановления Тв
   5. Скорость передачи информации
2. Физический тип и особенности
3. Стоимость

内存特性

* + - 1. 速度和时间比
         1. 访问时间：
         2. TD 内存周期时间（访问时间）Tcz
         3. 读取时间和写入时间
         4. тв冷却时间
         5. 信息传输速度
      2. 物理类型和特征
      3. 成本

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.

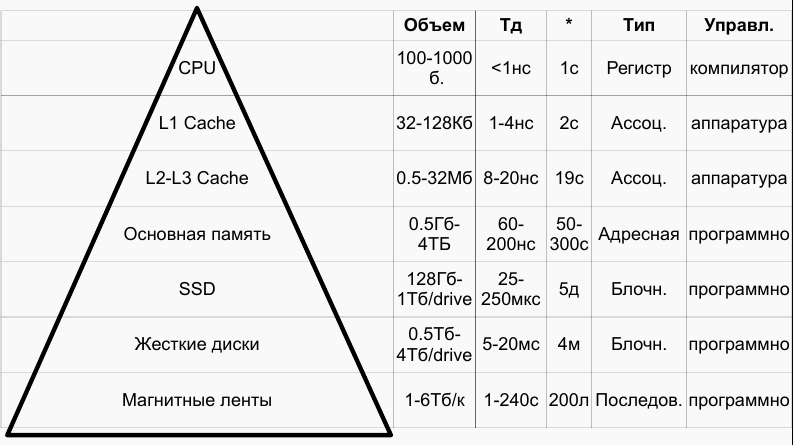
Время записи – интервал времени, необходимый для переписи содержимого шины данных в связанную с ней ячейку памяти.

存储设备具有许多质量指标来表征其信息和时间属性。

存储器的信息容量以由一定字节数组成的位数、字节数或字数表示。

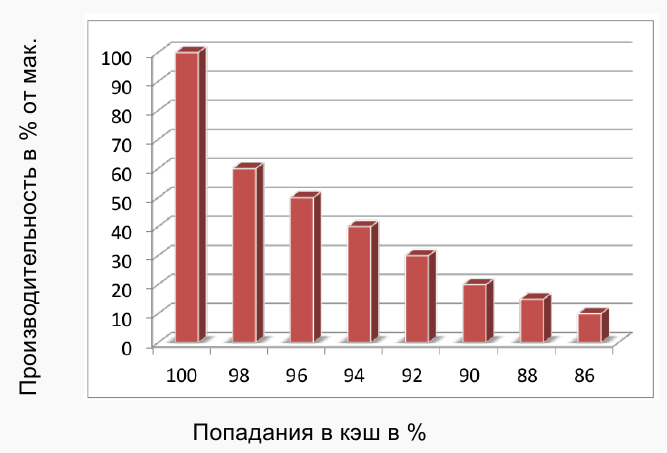
访问时间是从处理器在地址总线上设置所需小区的地址并通过控制总线发送读取或写入数据的请求到可寻址小区连接到数据总线的那一刻确定的时间间隔。

写入时间是将数据总线的内容写入其关联的内存位置所需的时间间隔。



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

关联内存，高速缓存。缓存未命中对性能的影响。



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.

处理器一直比内存快。处理器和内存正在并行改进，因此这种差异持续存在。随着越来越多的晶体管可以放置在芯片上，处理器设计人员利用这些优势创建了流水线和超标量架构，从而进一步提高了处理器的速度。内存设计人员通常使用新技术来增加容量而不是速度，这进一步加剧了这个问题。在实践中，这种性能不匹配会导致以下情况：处理器发出内存请求后，必须经过许多周期才能获得所需的字。内存越慢，处理器等待的时间越长，它必须通过的周期就越多。

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.

有两种方法可以解决这个问题。其中最简单的方法是在必要时开始从内存中读取信息，并继续执行指令，但如果任何指令在从内存中读取之前尝试使用单词，则处理器必须暂停。内存越慢，这个问题发生的频率就越高，工作损失就越大。例如，如果延迟为 10 个周期，则接下来的 10 个命令中的一个很可能会尝试使用尚未从内存中读取的单词。

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.

该问题的另一种解决方案是设计一台不暂停工作的机器，但确保编译器程序在从内存中读取之前不使用单词。然而，这在实践中并不是那么容易实现的。通常，当执行引导命令时，机器无法执行其他作，因此编译器被迫插入不执行任何作但占用内存空间的空命令。实际上，闲置的不是硬件，而是软件闲置，但性能损失是一样的。

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 А, то скорее всего после этого ей понадобится доступ к адресу, расположенному поблизости от А. 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.

其实，这个问题不是技术问题，而是经济问题。工程师知道如何构建与 CPU 一样快的内存，但它必须直接放置在 CPU 的芯片上（因为信息通过总线传输的速度非常慢）。在 CPU 芯片上放置大量内存会使其变大，因此更昂贵，即使成本无关紧要，可以构建的 CPU 大小仍然存在限制。因此，您必须在小容量的快速内存和大容量的慢速内存之间进行选择。我们更喜欢价格低廉、高速的大内存。具有高速运行的小型内存称为高速缓存（来自法语单词 cacher “隐藏”）。高速缓存背后的基本思想很简单：它包含最常用的单词。如果处理器需要一个字，它首先访问缓存。只有当单词不存在时，它才会访问主内存。如果缓存了很大一部分单词，则平均访问时间会显着减少。因此，成功或失败取决于缓存的单词数。人们早就知道程序不会随机访问内存。如果程序需要访问地址 A，则很可能需要访问位于 A 附近的地址。普通程序中的几乎所有命令（跳转命令和过程调用除外）都是从顺序内存位置调用的。此外，程序大部分时间都花在循环上，其中一遍又一遍地执行一组有限的指令。同样，在作矩阵时，程序可能会在转到其他内容之前多次访问同一矩阵。

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.

在连续引用记忆时，在一定时间内只使用一小部分记忆的事实称为局部性原理。这一原理构成了所有高速缓存系统的基础。这个想法是，当某个单词被调用出内存时，它会与相邻单词一起移动到缓存内存中，这允许您在下一个查询中快速引用下一个单词。如果一个单词被读写 k 次，计算机将需要对慢速主内存进行 1 次访问，对快速高速缓存进行 k-1 次访问。k 越高，整体性能越高。

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.

同时提高吞吐量和减少延迟的最有效技术之一是使用多个缓存块。主要技术是引入单独的指令缓存和单独的数据缓存（拆分缓存）。此缓存有几个优点。首先，作可以在每个缓存上独立启动，这将内存吞吐量加倍。这就是为什么我们在 Mic-1 微架构中需要两个独立的内存端口：每个缓存都有一个单独的端口。请注意，每个缓存都可以独立访问主内存。

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.

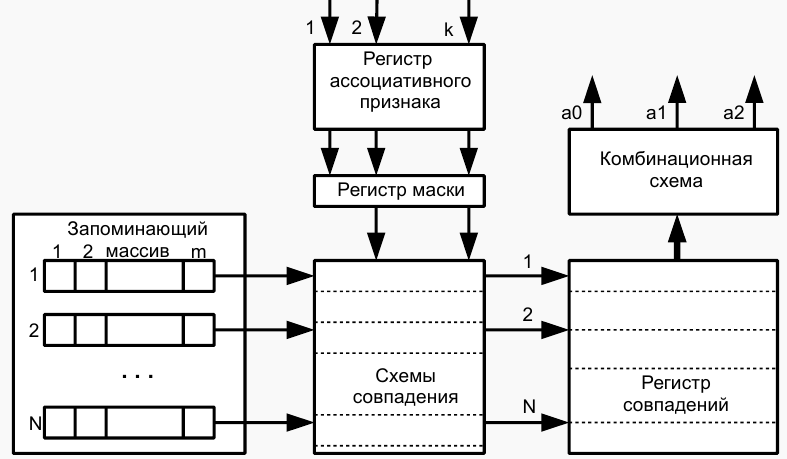
如今，许多内存系统比这些复杂得多。L2 高速缓存通常放置在拆分高速缓存和主内存之间。一般来说，可以有三个或更多缓存级别，因为需要更高级的系统。CPU 芯片上有一个用于指令的小缓存和一个用于数据的小缓存，通常在 16 到 64 KB 之间。还有一个二级高速缓存，它不是位于处理器芯片本身上，而是位于处理器芯片旁边的同一个块中。此缓存通常不分区，并且包含数据和命令的混合。它的大小范围从 512 KB 到 1 MB。L3 缓存与 CPU 位于同一板上，通常由几兆字节的静态 RAM 组成，这比动态主内存 RAM 快得多。通常，一级缓存的所有内容都在二级缓存中，二级缓存的所有内容都在三级缓存中。

**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](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2)», хотя последний термин чаще используется в программировании для обозначения структуры данных

关联存储器 （AM） 或关联存储器 （ASU） 是一种特殊类型的机器存储器，用于非常快速的搜索应用程序。它也称为内容可寻址内存、关联内存、内容可寻址内存或关联数组，尽管后一个术语在编程中更常用于指代数据结构

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.

在联想记忆中，元素不是按地址选择的，而是按内容选择的。让我们更详细地解释后一个概念。对于具有地址组织的内存，最小可寻址单元 （MAE） 的概念被引入为具有单个地址的数据的一部分。让我们为关联记忆引入一个类似的概念，我们将关联内存中的最小存储单元称为关联记忆 （StrAP） 字符串。每个 StrAP 包含两个字段：标签字段和数据字段。读取关联记忆的查询可以用单词表示如下：选择标签等于给定值的字符串。应该注意的是，使用这样的查询，可能出现以下三种结果之一：只有一行带有给定的标签;有几行带有给定的标签;没有一行带有给定标记。按要素搜索记录是数据库访问的典型活动，搜索数据库通常涉及关联搜索。要执行此类搜索，您应该查看所有记录并将指定的标记与每条记录的标记进行比较。当使用常规可寻址内存存储记录时，也可以这样做（很明显，这将花费相当多的时间，与存储的记录数量成正比！当硬件支持从内存中关联检索数据时，就会提到关联内存。



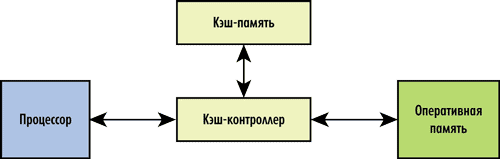
Принцип работы кэша процессора

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

现在我们已经弄清楚了处理器缓存的用途，让我们来看看缓存的基本原理，这些原理允许它实现其主要目的。

Кэш состоит из контроллера и собственно кэш­памяти. Кэш­контроллер 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 кэш­попаданием (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 кэш­промахе (cache miss) — тогда запрос данных переадресуется к оперативной памяти.

缓存由控制器和缓存本身组成。缓存控制器管理缓存的作，即将必要的数据从 RAM 加载到缓存中，并在必要时将处理器修改的数据返回到 RAM。在架构上，缓存控制器位于处理器和 RAM 之间（见图 1）。通过拦截对 RAM 的请求，缓存控制器确定缓存中是否存在请求数据的副本。如果存在这样的副本，则称为缓存命中，在这种情况下，从缓存中检索数据的速度非常快（比从 RAM 中检索快得多）。如果所需的数据不在缓存中，则称为缓存未命中，在这种情况下，数据请求将重定向到 RAM。

**

*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 кэш­контроллера 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.

为了获得最佳性能，缓存未命中应尽可能少发生（理想情况下，没有）。鉴于缓存在容量方面比 RAM 小得多，这并不容易实现。因此，缓存控制器的主要任务是将真正需要的数据加载到缓存中，并及时从中删除数据。重要的是要了解缓存始终是“已满”的，因为将部分缓存留空是不合理的。新数据只有通过逐出（替换）任何旧数据才能进入缓存。 缓存数据加载是基于缓存策略实现的，数据转储是基于替换策略实现的。

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

虚拟内存的用途和组织。段页组织。内存管理单元 （MMU）、转换缓冲区 （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.

虚拟内存是一种计算机内存管理技术，它允许作系统访问比计算机中实际安装的内存更多的内存。这是通过将数据放置在用作 RAM 的外部存储器的可用磁盘空间中来实现的。有必要了解，一些由于缺少 RAM 而无法放置在 RAM 中的程序现在将放置在 VZU 上，这相当于放置在 RAM 中。进水单元的使用非常方便，因为此时用户使用公共地址空间进行作，他不关心使用什么物理内存：外部或内部。

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.

在大多数现代作系统中，虚拟内存是使用分页来组织的。OP 分为多个页面：固定长度的内存区域，这是分配的最小内存单位。进程使用包含页码和页内偏移量的虚拟内存地址访问内存。如果从 LO 卸载页面，作系统将从硬盘交换页面。当请求内存分配时，作系统可以将长时间未访问的页面“刷新”到硬盘。关键数据（例如正在运行和正在运行的程序的代码、系统内核的代码和内存）通常位于 RAM 中。

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.

段组织是一种组织虚拟内存的机制，其中虚拟空间被划分为任意大小的部分——段。对于每个段，就像一个页面一样，可以将其访问权限分配给用户及其进程。加载进程时，一些段被放置在 RAM 中，一些段被放置在磁盘内存中。一个程序的段可以占用 RAM 中的不相邻区域。这种内存分配方法的缺点是分段级别的碎片和地址转换比分页慢。

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.

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

网络技术、计算机网络的概念、计算机网络的分类。消息和包。开放系统之间的交互模型。

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

计算机网络是一组计算机，无需中间数据载体即可在它们之间进行信息交换。

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

计算机网络是一个由相互连接的硬件和软件组件组成的复杂系统。

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

软件组件由网络作系统和网络应用程序（邮件程序、网络数据库）组成。

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

¾   рабочие станции;

¾   серверы сети;

¾   коммуникационные узлы.

连接到网络的所有设备（即硬件组件）可分为三个功能组：    工作站;    网络服务器;    通信节点。

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.

工作站是连接到网络的 PC，网络用户在其上执行其工作。每个工作站处理自己的本地文件并使用自己的作系统。

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.

网络服务器是连接到网络并为网络用户提供某些服务的计算机，例如，存储公共数据、处理对 DBMS 的请求、打印任务、远程处理任务等。

К коммуникационным  узлам относятся следующие устройства:

¾   повторители;

¾   коммутаторы (мосты);

¾   маршрутизаторы;

¾   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:

***通信节点包括以下设备：***

***中继 器;***

***开关（桥）;***

***路由器;***

***网关。***

***这些是将不同网络相互连接所需的设备。 创建计算机网络是为了为任何网络用户提供对任何网络资源的潜在访问。***

***网络技术分类：***

******

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

消息切换是将信息划分为消息，每个消息由一个标头和信息组成。

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

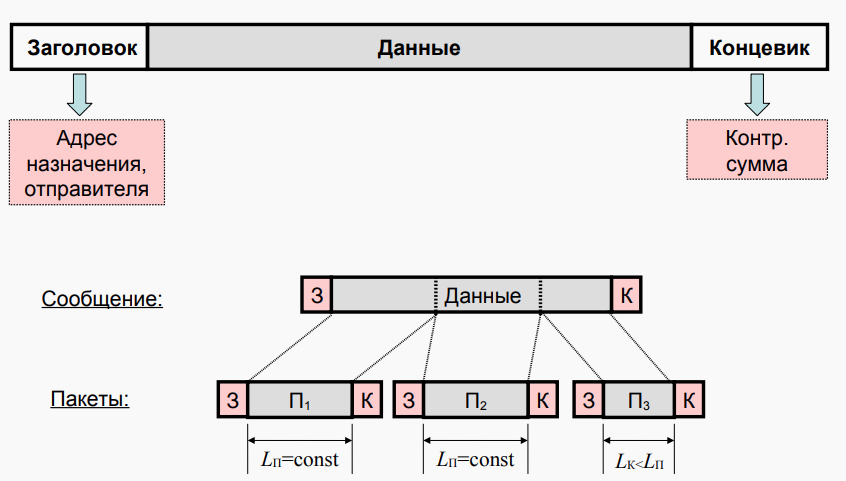
这是一种通信方法，其中通过通信节点将消息按顺序传递到消息头中指定的地址来创建逻辑通道。

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

分组交换是一种特殊的网络节点交换方式，专为计算机流量（脉动流量）的最佳传输而创建。

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](https://sysadmin.ru/docs/network/ethernet" \t "_blank)**). 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.

当数据包被交换时，网络用户传输的所有消息都会在源节点上分解成相对较小的部分，称为数据包。应该澄清的是，消息是逻辑上完整的数据片段 - 传输文件的请求、对包含整个文件的请求的响应等。消息可以有任意长度，从几字节到几兆字节。相比之下，数据包通常也可以具有可变长度，但在狭窄的范围内，例如 46 到 1500 字节 （EtherNet）。每个数据包都提供了一个标头，该标头指定将数据包传送到目标节点所需的地址信息，以及目标节点将用于组装消息的数据包号。



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

它由 7 个级别组成，每个级别执行特定的角色和任务。让我们从下往上看一下每个级别的作用：  
  
**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.

1）物理层：定义数据传输的方法、使用什么介质（电信号的传输、光脉冲或无线电以太）、电压电平、二进制信号的编码方法。  
  
**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.

2）数据链路层：接管局域网内寻址的任务，检测错误，验证数据的完整性。如果您听说过 MAC 地址和以太网协议，那么它们就位于此级别。  
  
**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.

网络层：该层负责组合网络部分并选择最佳路径（即路由）。每个网络设备在网络上必须有一个唯一的网络地址。我想很多人都听说过 IPv4 和 IPv6 协议。这些协议在此级别上工作。  
  
**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).

传输层：该层接管传输功能。例如，当您从 Internet 下载文件时，该文件将以段的形式发送到您的计算机。它还引入了指定特定服务的目的地所需的端口概念。TCP（连接到）和 UDP（无连接）协议在此级别工作。  
  
**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.

会话层：此层在建立、管理和终止两台主机之间的连接中的作用。例如，当您在 Web 服务器上打开页面时，您不是该页面上的唯一访问者。  
  
**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.

表示层：它将信息结构化为应用层的可读形式。例如，许多计算机使用 ASCII 编码表来显示文本信息或使用 jpeg 格式来显示图形。  
  
**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.

应用层：这可能是每个人最容易理解的级别。正是在这个级别上，我们习惯于工作的应用程序——电子邮件、通过 HTTP 协议的浏览器、FTP 等。

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

TCP/IP模型：传输介质、链路层、网络层。寻址、转发和路由数据包。

Модель TCP/IP:

TCP/IP:

* Прикладной уровень
* Транспортный уровень
* Сетевой уровень
* Канальный уровень
* Среда передачи: витая пара, коаксиальный кабель, оптика, радиосигналы
* 应用层
* 传输层
* 网络层
* 通道层
* 传输介质：双绞线、同轴电缆、光学器件、无线电信号

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 5е: 100 Мбит/c (2 пары), 1Гбит/c на (4пары)

– Категория 6: 10 Гбит/c (55м)

– Категория 7а: 40Гбит/c (50м), 100Гбит/c (15м)

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

– ST (Straight Tip)

– SC (Standard Connector)

– LC (Lucent Connector)

– Лазер находится в SFP (Small Plugin Factor)

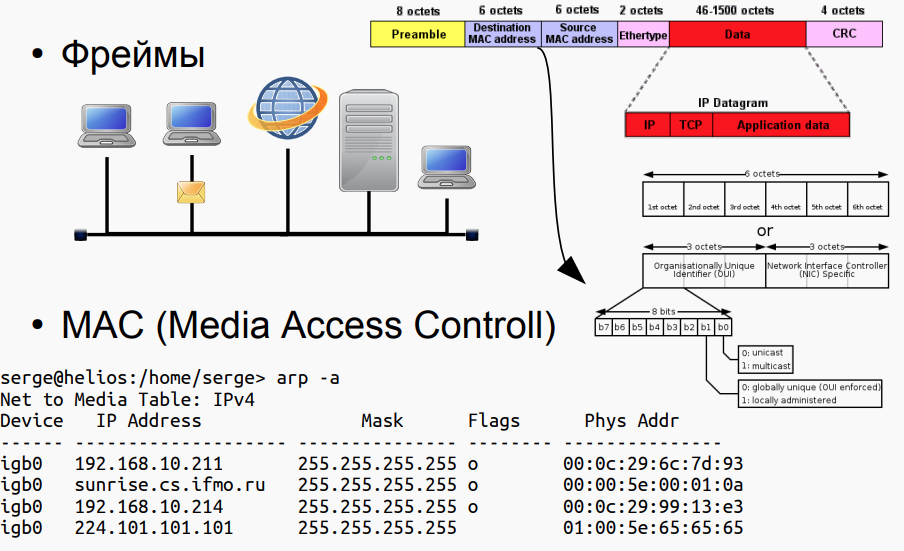
– ~500 м (Multi-mode fiber), ~80км (Single Mode)

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

– 2.4, 5, 60 GHZ

– До 15 Гбит/c

Канальный уровень Ethernet:



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.

以太网帧定义通过网络传输的数据的消息格式。消息格式包含多个信息字段，包括要传输的数据。数据块被定义为将发送的实际数据，可以包含 46 到 1500 字节的二进制信息。数据块的长度被定义并包含在消息中，作为接收器的字段，以确定消息的哪一部分是数据。

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.

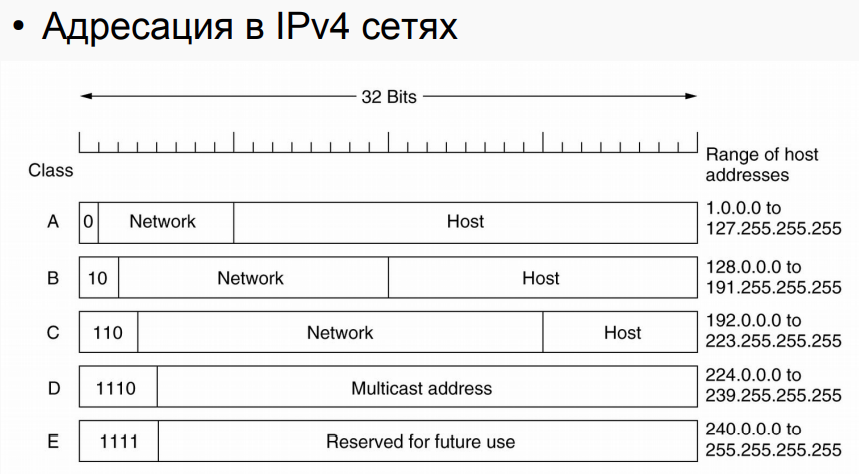
MAC 地址是一个 6 字节的二进制拨号号码，包括节点的源和目标信息。MAC地址包含在每条消息中并通过网络传输，以太网网络上的每个节点都有一个唯一的MAC地址。MAC（媒体访问控制）地址是分配给每个计算机网络设备的唯一标识符。

**Сетевой уровень**предназначен для 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.

网络层旨在通过许多物理网络（通信线路）将数据包从源传送到目的地。网络层确保每个数据包从其源点传送到最终目的地。一些网络层职责包括逻辑寻址和路由。

**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.
* 网络层旨在将单个数据包从源主机传送到目标主机。 互联网协议 （IP）。IP 是根据 TCP/IP 协议使用的传输机制。这是一项不可靠、无连接的服务，但初衷是好的。“善意”一词。表示IP不提供错误检查或最佳路由选择。IP考虑到主层的不可靠性，“尽一切努力”将信息传输到目的地，但没有保证。IP 以称为数据报的数据包的形式传输数据，每个数据包都是单独传输的。数据报可以传播不同的路线，并且可能以不同的顺序到达或重复。IP 不保留路由的顺序和列表，并且无法在到达目的地后更正数据报。IP 的有限功能不能被视为弱点。IP仅提供传输功能。用户可以添加此应用程序所需的工具，从而确保最大效率。



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.

路由是确定/计算通信网络中数据最佳路由的过程。还有第二个定义 - 将数据包从发送方传输到接收方。 路由功能可以通过以下方式执行： 硬件是路由器。允许您处理大型数据流并且运行速度更快的最佳选择。 配置了具有多个网络接口的计算机，并在其上安装了专用和定制软件。如果配置不会太复杂，通常会使用它。 路由表是驻留在路由器或特殊配置的计算机上的电子表格或数据库文件。它描述了如何将目标地址映射到要通过该接口将数据发送到下一个路由器。

**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.

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

TCP/IP 模型：地址分配 （DHCP）、名称服务、传输层和应用程序层。

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.

DHCP 是一种用于自动将 IP 地址分配给客户端的协议。它广泛应用于现代网络。在本文中，我们将考虑协议的作原理、DORA 流程、主要选项和其他方面。DHCP 是 TCP/IP 模型的应用层协议，用于将 IP 地址分配给客户端。这源于它的名字——动态主机配置协议。可以手动将 IP 地址分配给每个客户端，即本地网络上的计算机。但在大型网络中，这是非常费力的，而且本地网络越大，配置时出错的概率就越高。因此，创建了 DHCP 协议来自动分配 IP。

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.

最初，客户端处于初始化状态 （INIT），没有自己的 IP 地址。因此，它向本地网络上的所有设备发送 DHCPDISCOVER 广播消息。DHCP 服务器位于同一本地网络上。例如，DHCP 服务器是路由器或交换机，也有专用的 DHCP 服务器。

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.

DHCP 服务器会使用建议响应查找，它会告诉您可能适合客户端的 IP。IP 是从管理员设置的可用地址的 SCOPE 中分配的。

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](https://tools.ietf.org/html/rfc7719) 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](http://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](https://howdns.works/ep1/).  
  
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);

} }

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

I/O 接口。外部设备控制器。标准化水平，与系统总线的接换周期。控制器寄存器。

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

接口：

● 定义具体的交换细节

– 频率、传输通道集、编码方法、命令、数据表示、数据集和序列......

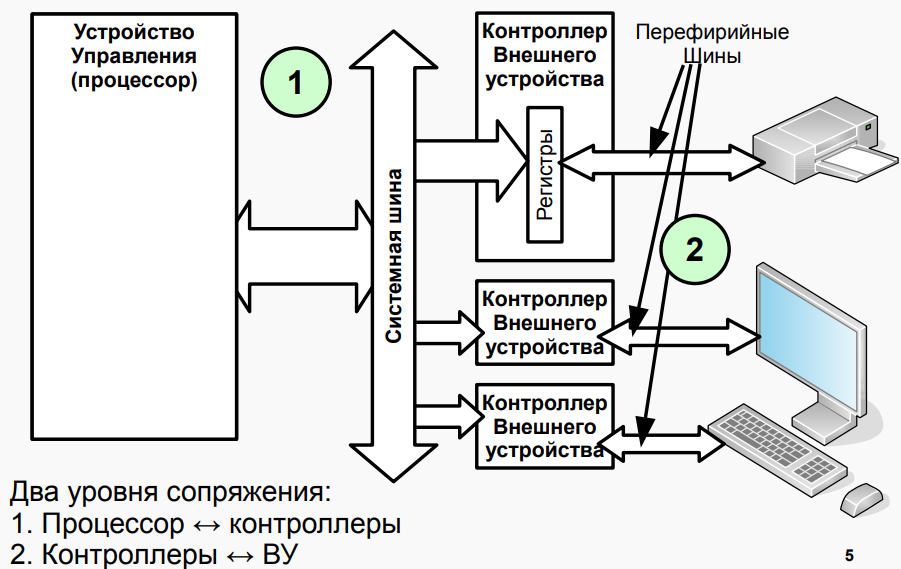
● 硬件和/或软件实施

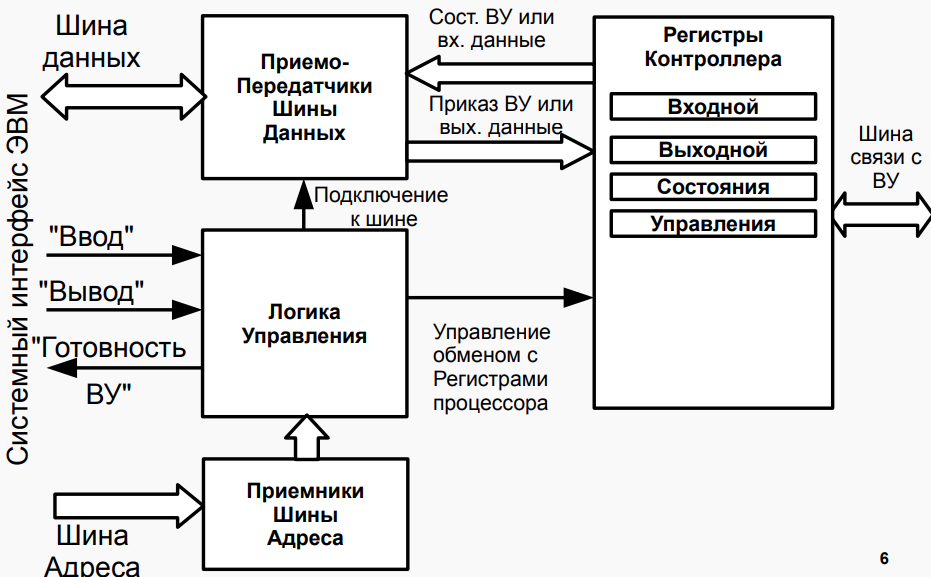
● 需要精确的规范和/或标准化 – 交换各方必须明确解释交换的细节 接口标准化水平

● 逻辑连接

● 信号物理参数

● 设计特点





Контроллеры внешних устройств

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

任何外部设备与微型计算机的连接都是通过 VU 控制器进行的。组织控制器的方法由两个因素决定：

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:

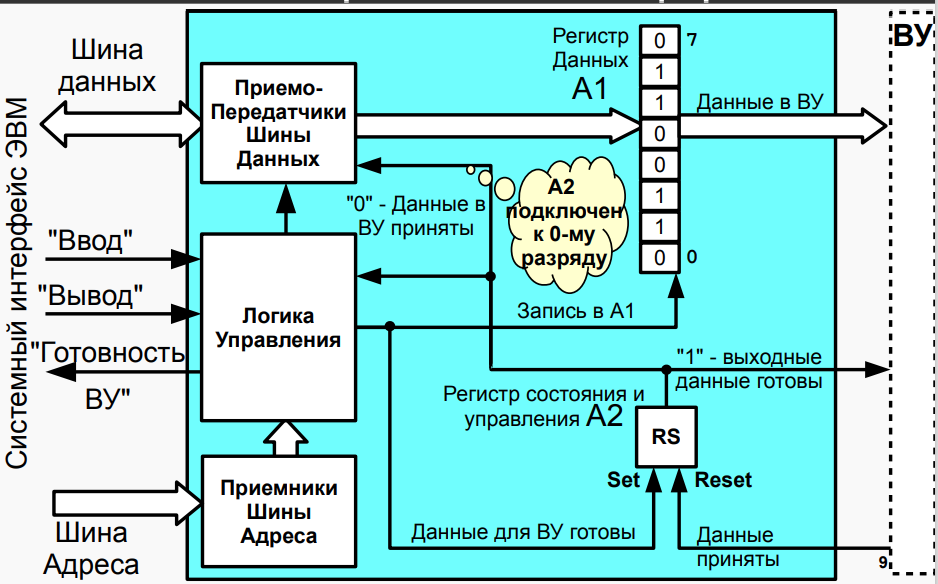
* Связь с ВУ по стандартному параллельному (ИРПР) каналу передачи данных;
* Связь с ВУ по стандартному последовательному (ИРПС) каналу передачи данных;
* Преобразование информации из аналоговой в цифровую с заданной точностью;
* Преобразование информации из дискретной формы представления в аналоговую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

还有可编程控制器，其作模式由特殊的微机命令设置或由与 VU 的交换程序确定。此类控制器必须配置为特定的交换模式（同步、异步、带或不带中断信号）。控制器使用特殊方法以编程方式配置。命令

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

并行数据传输。并行发送和接收控制器。



Параллельная передача данных в ВУ под управлением программы асинхронного обмена:

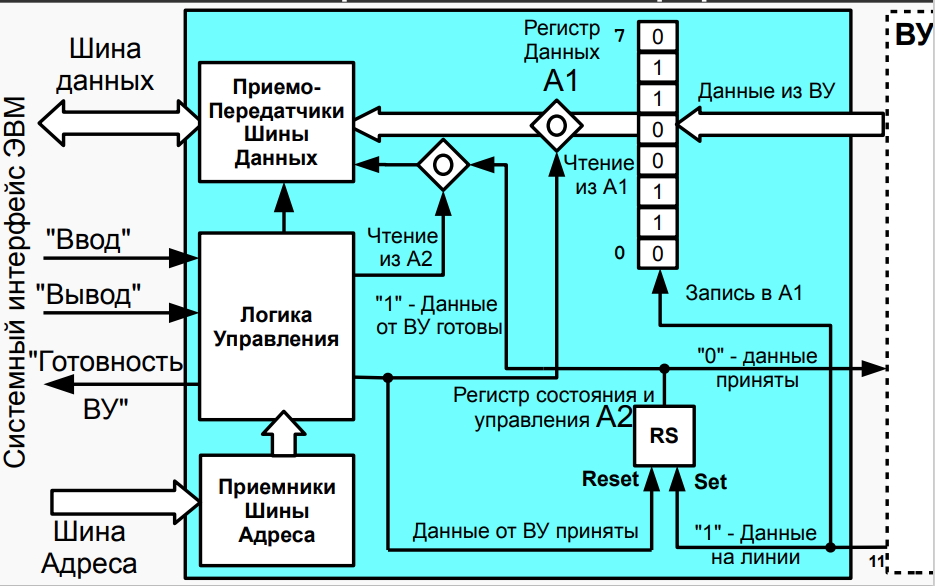
1. Процессор микроЭВМ проверяет готовность ВУ к приему данных
2. Если ВУ готово к приему данных (логический 0 в регистре А2), то данные передаются из шины данных системного интерфейса в регистр данных А1 контроллера и далее в ВУ. Иначе повторяется пункт 1.

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»).

在 VU 通信总线中，使用 2 个控制信号。生成控制信号“输出数据就绪”并从驾驶执照接收。信号“数据接收”时，控制器使用单位可寻址状态和控制寄存器 A2。同时从系统的数据总线写入下一个字节的数据。接口到控制器 A1 的可寻址数据寄存器，将逻辑单元写入状态和控制寄存器（生成控制信号“输出数据就绪”）。

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.

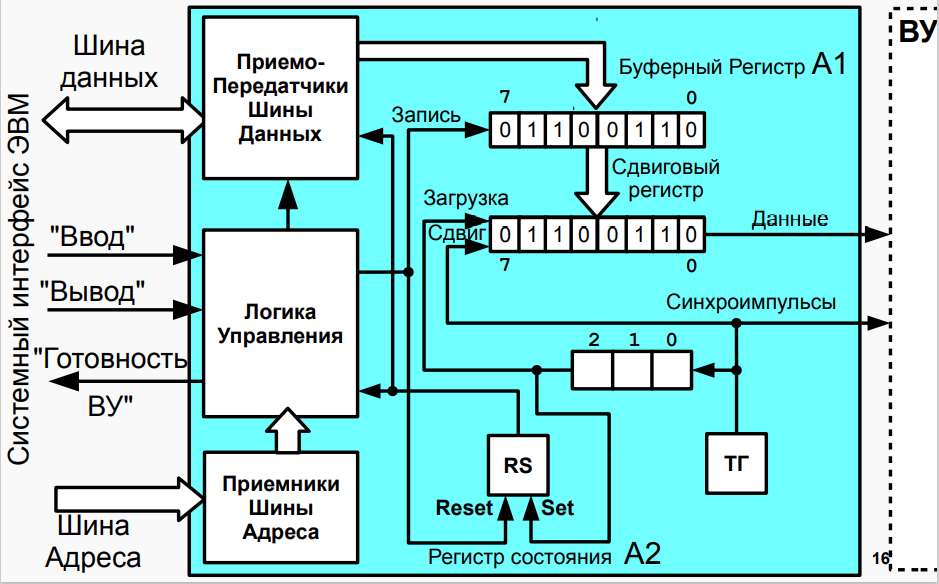
ВУ записывает в регистр данных контроллера А1 очередной байт данных и управ. сигналом «Данные от ВУ готовы» устанавливает в единицу регистр состояния и управления А2. При этом формируется:

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».

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

同步串行接口。串行和接收控制器。



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.

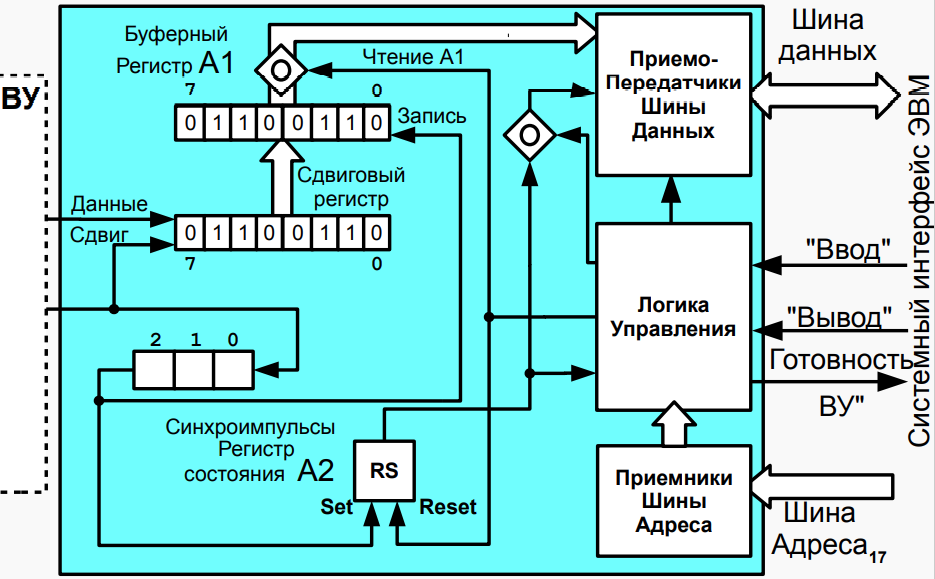
A1 控制器的 8 位缓冲区寄存器 - 用于临时存储一个字节的数据，直到它被加载到移位寄存器中。当 A2 状态寄存器中有 1 时，一个字节的数据将写入缓冲区寄存器。该寄存器的内容通过数据总线的一条线路传输到处理器，并用于生成控制。信号“VU准备就绪”。当将另一个字节写入寄存器 A1 时，寄存器 A2 为零。

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.

在移位寄存器中，数据从并行格式转换为串行格式并传输到通信线路。在下一个时钟脉冲上，移位寄存器的内容向右移动 1 位，下一位的值被赋予“数据”通信线。在移位的同时，时钟脉冲沿同步线传输。

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.

传输到线路的时钟信号数（传输的位数）由时钟计数器计数。一旦其内容为 7（传输 8 位信息），就会生成“加载”控制信号，并将缓冲区中的下一个字节写入移位寄存器。设置为 1 个状态寄存器。对于下一个时钟脉冲，计数器将重置为 0，并且下一个从移位寄存器向通信线发出 8 位的周期将开始。



Буферный регистр контроллера А1 - для временного хранения байта , поступившего из сдвигового регистра. Чтение байта данных из буферного регистра происходит при наличии 1 в регистре состояния А2.

A1控制器的缓冲区寄存器用于临时存储从移位寄存器接收的字节。当 A2 状态寄存器中存在 1 时，从缓冲区寄存器读取一个字节的数据

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.

来自串行码链路的数据使用移位寄存器转换为并行。控制器中的“数据”线连接到移位寄存器的串行输入，“同步”线连接到控制器。移位输入和时钟计数器输入。沿“同步”线的时钟脉冲将移位寄存器的内容向左移动 1 位，并将“数据”行的下一位数据写入该寄存器的最低位。同时增加 1 个时钟计数器。一旦它等于 7，就形成了一个板。“写入”信号，移位字节中的一个字节被写入缓冲区寄存器。设置为 1 个状态寄存器。

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

当一个字节的数据从缓冲区寄存器传输到数据总线时，状态寄存器被归零（即，在移位寄存器中接收到下一个字节的信息）。

1. 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:

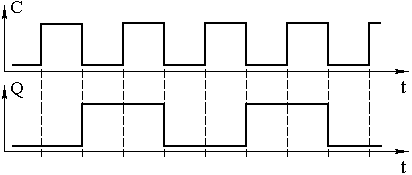
1. ЭВМ не выполняет никаких полезных действий во время ожидания момента готовности ВУ.

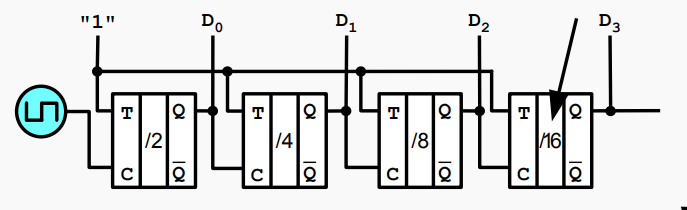
Принципы деления частоты

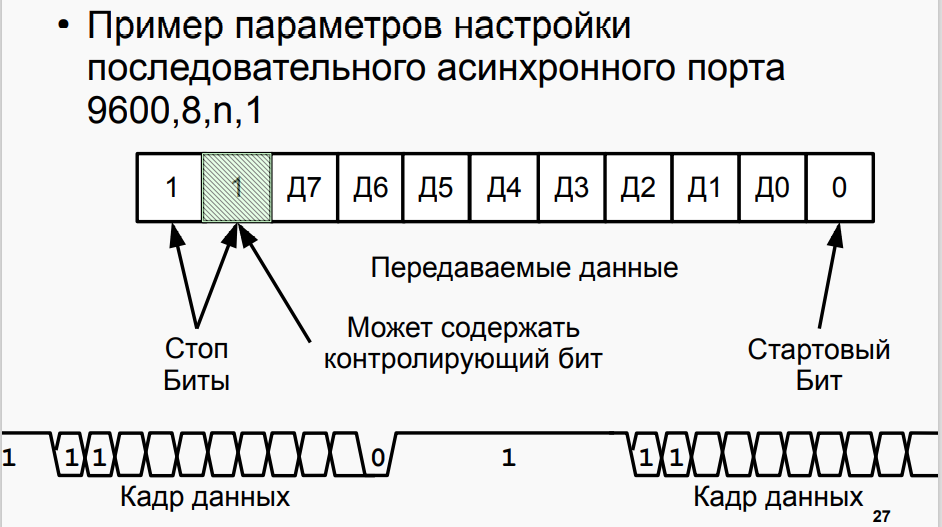
При асинхронной передаче данных, со временем может происходить рассинхронизация генераторов тактов передатчика и приёмника, в результате чего данные могут быть переданы с искажениями, или не быть переданы вовсе.

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.

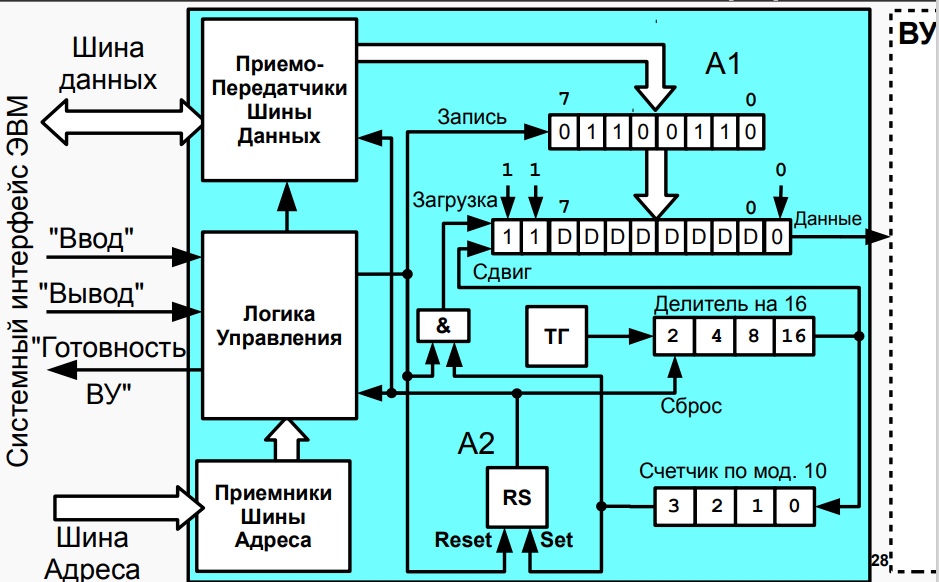






1. Контроллер передачи асинхронного последовательные интерфейса.

异步串行接口传输控制器。

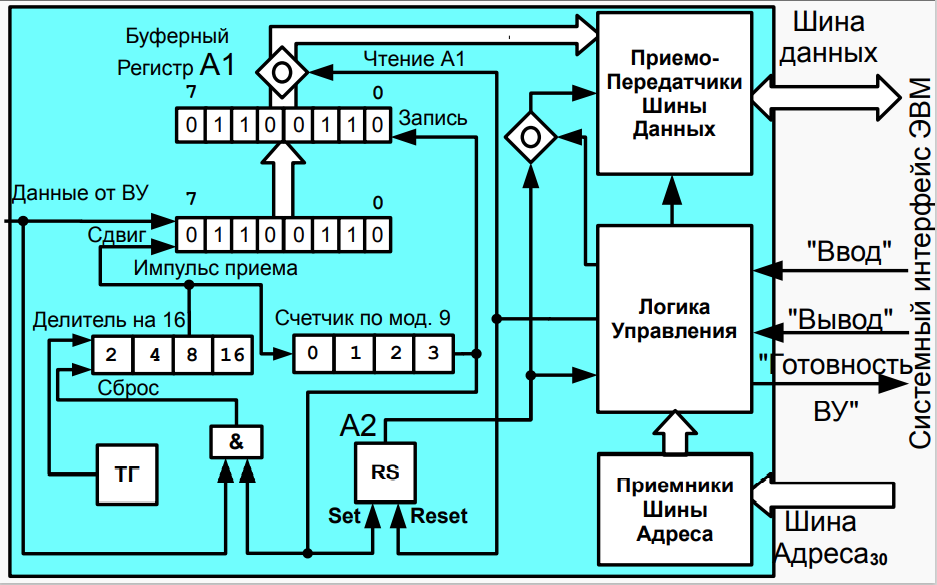


Процесс передачи

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

将前一个字节的数据传输到 A2 状态寄存器后，写入 1，通知处理器控制器已准备好接收下一个字节的数据，并通过通信线路将其传输到 VD。它还禁止从产生移位脉冲的电路中产生脉冲 - 时钟发生器的分频器 16（根据 mod 16 的计数器）。剪切脉冲计数器（mod 10计数器）处于零状态，其单元输出信号被传输到栅极I，为加载移位寄存器准备信号生成电路。 处理器执行“输出”命令，在数据总线上设置传输的字节，并生成“输出”系统接口的控制信号。 在“输出”信号处，控制器将传输的字节写入缓冲区寄存器A1，复位状态寄存器A2，并在门I上生成“负载”信号。 传输的位被重写到移位寄存器的位 1..8 中，零（起始位）被写入位 0，停止位被写入位 9 和 10 单位。 “复位”信号从分频器中移除，它开始积累时钟发生器的脉冲，并在接收到第十六个时钟脉冲的那一刻，触发移位脉冲（这就是频分的实现方式）。 数据总线保持 0（起始位值），直到达到第一个移位脉冲（1 位传输时间）。剪切脉冲将改变剪切脉冲计数器的状态，并将传输数据字节的第一个信息位覆盖到移位寄存器的零位。该位的值将保留在数据线上，直到下一个移位脉冲。 其余的信息位以同样的方式传输，第一个停止位，最后是第二个停止位，传输时，移位脉冲计数器将再次设置为零。这将导致 1 写入 A2 状态寄存器。来自A2寄存器输出的单个信号将阻止移位脉冲的形成，并通知接收新字节数据的准备情况。 下一帧（起始位、信息位和 2 停止位）的传输完成后，第二个停止位的值保持在数据传输线上 - 一个

1. 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:
   1. Содержимое сдвигового регистра будет переписано в БР
   2. В РС А2 запишется 1 и он будет информировать процессор об окончании приема очередного байта
   3. Вентиль И подготовится к вы работке сигнала «Сброс»(он сформируется после прихода первого стопового бита).
   4. 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

转移流程 “数据”线上有一个单元，禁止时钟发生器分频器的作。 当在“数据”线上检测到零信号（将停止位更改为起始位）时，“复位”信号从分频器中移除，它开始累积时钟频率发生器的脉冲。 当计数器累积值为8（位传输时间的一半）时，它将输出一个信号，传送到移位寄存器和移位脉冲计数器的输入端。（这减少了数据损坏的机会。 随后的移位发生在 16 个时钟脉冲通过后。 当帧的第 9 位（接收到第 8 位）进入移位寄存器时，起始位将移出移位寄存器，因此，信息字节将被放置在移位寄存器中。此时，剪切脉冲计数器将归零，并在其输出端生成单个信号，根据该信号： 移位寄存器的内容将在 BR 中被覆盖 在 PC A2 中，将写入 1，它将通知处理器接收下一个字节的结束时间 阀门 I 将准备“复位”信号的运行（它将在第一个停止位到达后形成）。 收到就绪信号（PC A1 中的 2 个信号）后，处理器将执行“Enter”命令。同时，生成系统接口的“输入”信号，通过该信号将接收到的数据字节从 BR 发送到处理器（“读取”信号），并重置 A2 RS

1. Организация прямого доступа к памяти. Контроллер ПДП

直接内存访问。直接内存访问控制器

Этот тип обмена реализуется полностью аппаратно и управляется контроллером ПДП.

这种类型的交换完全在硬件中实现，并由直接内存访问控制器控制

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.

直接访问内存的特点

能够将程序从输入设备引导到微型计算机的主存储器中。

提供在微型计算机中使用高速外部存储器的能力。

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.

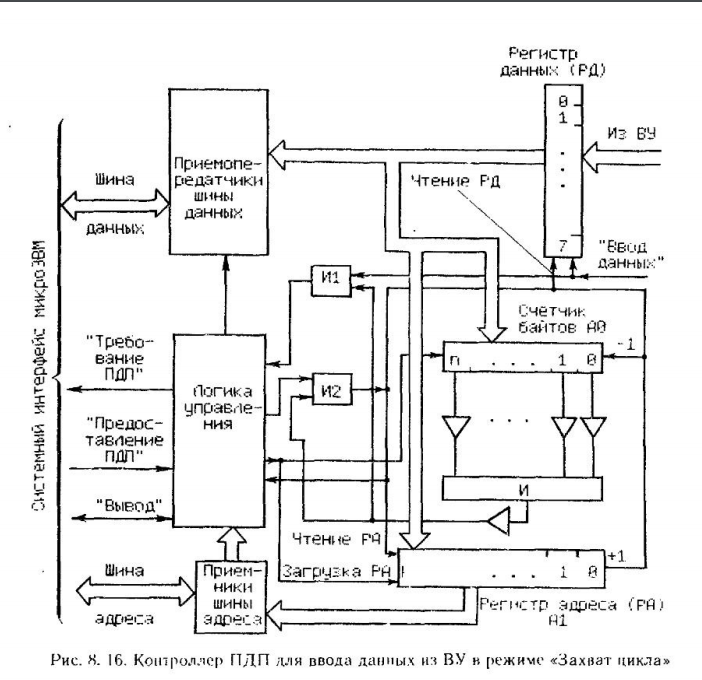
为了节省资源，PDP控制器没有自己的资源，而是连接到计算机系统接口的数据总线（SD）和地址（SH），这使得PDP控制器和处理器无法同时使用总线。

Эта проблема решается двумя способами:

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

1. 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.
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