ВОПРОСЫ К ЭКЗАМЕНУ ПО ОПД

----------Саша, Женя

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

Две формы представления информации:

- Непрерывная (аналоговая)
  Характеризует процесс, который не имеет перерывов и может изменяться в любой момент времени и на любую величину (запись звука на магнитную ленту, напряжение) “*в данном примере от звука зависит напряжение(ведь оно его записывает, делается это непрерывно по понятным причинам”*
- Прерывистая (цифровая, дискретная)
  Цифровой сигнал может изменяться лишь в определенные моменты времени и принимать заранее обусловленные значения (текст, числа).

Системы счисления, используемые в ВТ:

- Двоичная
  0 или 1 для каждого бита, либо есть сигнал, либо его нет, очень удобная
- Восьмеричная
  от 0 до 7. Широко использовалась в программировании и компьютерной документации, но позднее была почти полностью вытеснена шестнадцатеричной.
- Десятичная
  “Не до конца” удобная для машины, однако понятная для человека, при написании кодов, мнемоник и вообще выводов(по типу калькулятора)
- Шестнадцатеричная
  от 0 до F. Удобна, поскольку минимальной адресуемой единицей памяти является байт (8 бит), значения которого удобно записывать двумя 16-ричными цифрами. Используется в RGB.
- Двоично-десятичная
  Используется для хранения десятичных чисел в памяти компьютера.
  Десятичная цифра задается 4 битами (от 0000 до 1001).
  +Удобно для вывода чисел на индикацию.
  +Для дробных чисел при переводе в десятичный формат не теряется точность.
  -Требует больше памяти.
  -Усложнены арифметические операции.

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

Вспоминаем дискретку: существуют форматы, такие как Ф1, Ф2 и Ф3, которые позволяют представить числа с фиксированной запятой. Общий вид таков: один бит под знак, 7-8 бит под порядок, оставшиеся биты под мантиссу.

Прямой код представляет собой одинаковое представление значимой части числа для положительных и отрицательных чисел и отличается только знаковым битом.

$$6_{10} = 0110_2$$
$$-6_{10} = 1110_2$$

Обратный код для положительных совпадает с прямым, а для отрицательных образуется из прямого кода положительного числа путем инвертирования разрядов.

$$6_{10} = 0110_2$$
$$-6_{10} = 1001_2$$

Дополнительный код для положительных совпадает с прямым, а для отрицательных образуется путем добавления 1 к обратному коду.

$$6_{10} = 0110_2$$
$$-6_{10} = 1010_2$$

Формирование битовых признаков:

- Перенос (C, CF)
  Единица выносится за пределы разрядной сетки при сложении или занимается из этих пределов при вычитании.
  Плохо для беззнаковых чисел.
- Переполнение (V, OF)
  Единица выносится в знаковый разряд числа при сложении или занимается из этого разряда при вычитании.
  Плохо для знаковых чисел.
- Отрицательный результат (N, NF)
  Единица в знаковом бите.
- Нуль (Z, ZF)
  Все разряды равны нулю

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

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

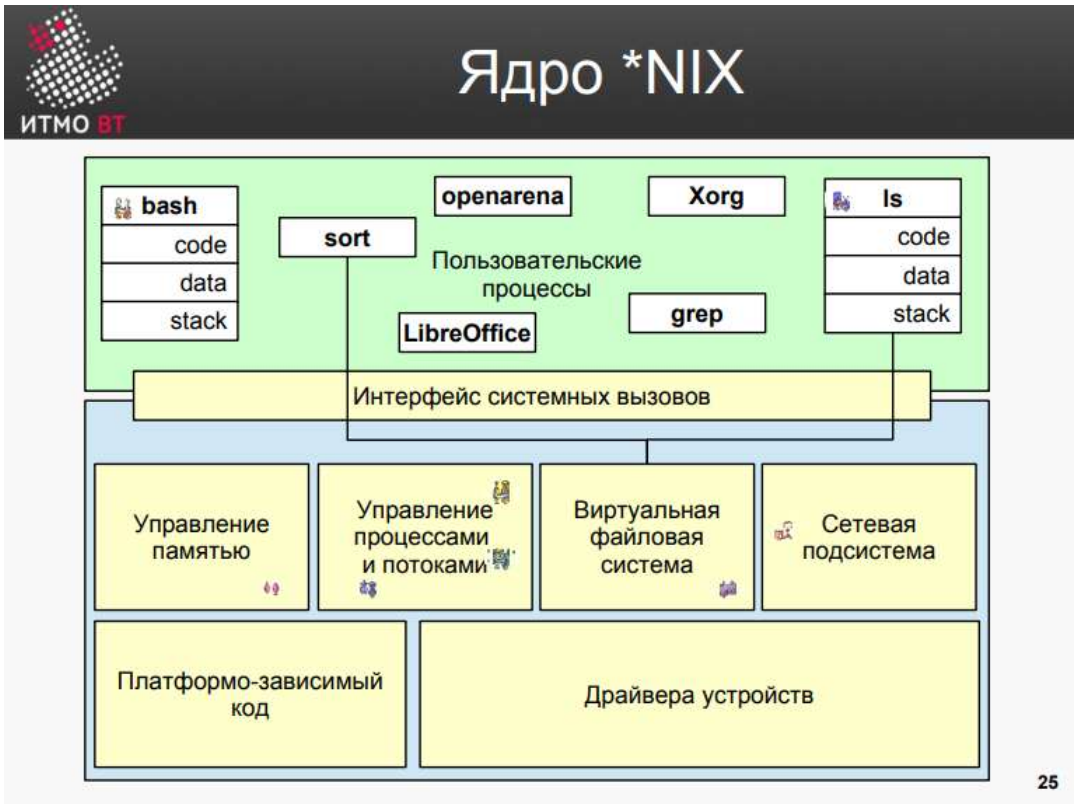
5. Структура и принцип функционирования ЭВМ. Порядок функционирования простого процессора на примере калькулятора.

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

UNIX - семейство переносимых/кроссплатформенных, многозадачных и многопользовательских операционных систем. Характеризуются модульным дизайном, в котором каждая задача выполняется отдельной утилитой, взаимодействие осуществляется через единую файловую систему, а для работы с утилитами используется командная оболочка.

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 межпроцессного взаимодействия в виде файлов
- Использование конвейеров из нескольких программ, каждая из которых выполняет одну задачу



Ядро - набор подсистем и программ, управляющие оборудованием и программами пользователя. В ядре выполняются служебные операции ОС.

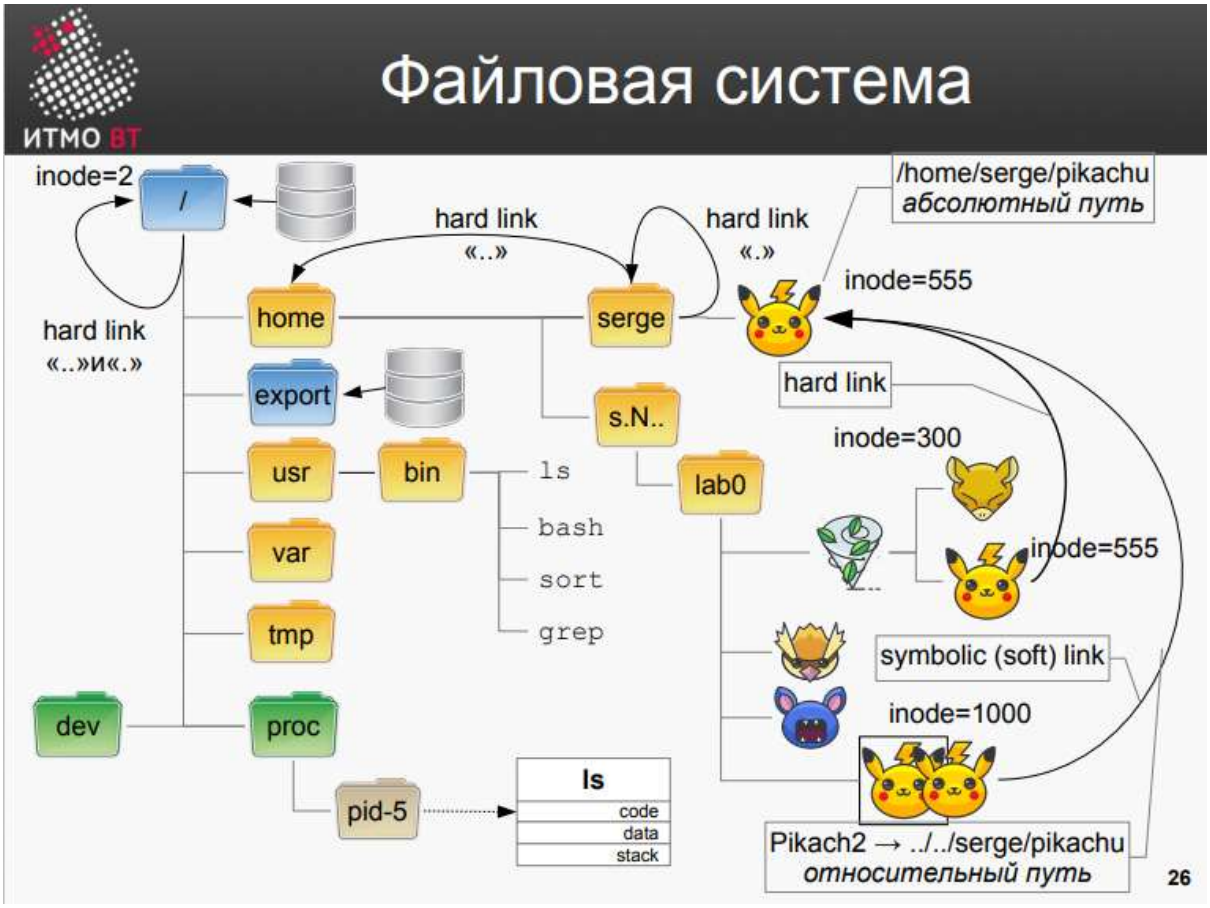
Взаимодействие программ пользователя происходит с помощью интерфейса системных вызовов.

Интерфейс системных вызовов - стандартизированное средство взаимодействия программы пользователя и ядра: доступ к сервисам ядра можно получить только через этот интерфейс.

Ядро системы разработано таким образом, что его легко можно приспособить практически под любой микропроцессор.

Основные подсистемы:

- Управление памятью
- Управление процессами и потоками
- Виртуальная файловая система
- Сетевая подсистема
- Платформо-зависимый код
- Драйвера устройств



По философии UNIX **всё есть файл**
Все файлы располагаются в файловой системе, представляющей собой дерево, промежуточные вершины которого соответствуют каталогам, а листья -

файлам и пустым каталогам. Каждый каталог и файл имеет уникальный полный путь.
Абсолютный путь - это путь от корня файловой системы.
Относительный путь - это путь от текущего рабочего каталога.
“.” - ссылка на текущий каталог, **“..”** - ссылка на родительский каталог

В Unix все является файлом кроме потоков, процессов и ядра.

Существуют ссылочные файлы:

- **Жесткие ссылки**. Имеют такой же inode (Index-node, уникальный номер файла), как у исходного файла
- **Символические ссылки**. Содержат только путь к файлу. При удалении исходного файла ссылка останется, но будет указывать никуда.

В **inod’e** хранится тип файла, права, время модификации/создания файла и другая служебная информация (метаданные).

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

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

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

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

----------Миша

11. Организация массивов данных. Режимы адресации. Цикл выборки адреса и операнда БЭВМ.
Массивы в БЭВМ организованы просто как n значений, которые подряд записаны в ячейки памяти. Обычно используют какую-то переменную, со значением адреса первого или последнего элемента массива (ссылка на массив). В таком случае удобно обходить все значения массива, используя косвенную автоинкрементную/декрементную адресацию.
В БЭВМ есть не так много видов адресации:

1. Абсолютная, когда мы напрямую указываем адрес после кода операции, в 11 бите 0

Следующие команды с относительной адресацией т.е будет обращение к ячейке памяти относительно текущего адреса, в 11 бите 1. Будут перечислены по типам

2. Косвенная, когда мы обращаемся по адресу, который лежит в ячейке, к которой мы обращаемся
3. Прямая относительная, когда мы просто работаем с ячейкой, которая смещена относительно текущего адреса
4. Прямая загрузка операнда, когда аргумент команды напрямую загружается в младший байт

Вот все виды адресации с их циклами и циклом выборки операнда, тут думаю пояснения не нужны

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

Где здесь AF и OF?

23

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

Для управления вычислительным процессом в БЭВМ используются команды ветвления, команды LOOP, CMP и JUMP. Команды ветвления работают на основе флагов, и при определенных значениях этих флагов изменяют текущий адрес на тот, который лежит в аргументе команды. Очень часто нужна какая-то операция сравнения одного числа с другим. Для этого в БЭВМ есть команда CMP, которая выставляет флаги по результату вычитания из AC, переданного командой значения. После CMP ставят нужную команду ветвления. Это полный аналог привычной нам конструкции if/else.

Вот все команды ветвления, который есть в базовой ЭВМ

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

Для организации циклов, в основном используются команды LOOP и JUMP. LOOP чем-то похож на команду ветвления. Он уменьшает на 1 значение, переданной ему ячейки, и если оно становится не положительным, то пропускает следующую команду. Команда JUMP безусловно меняет текущий адрес на переданный. Используя эти две команда, идущие подряд, организовывается цикл.
Цикл исполнения LOOP:

~0 + DR → DR
~0 + DR → BR; DR → MEM(AR)
if BR(15) = 0 then GOTO INT @ C4
IP + 1 → IP

Мы уменьшаем аргумент на 2, и проверяем не отрицательный ли он. Это сделано так, потому что мы не можем проверить по флагам стал ли аргумент не положительным, т.к. флаги при вышеописанных операциях не выставляются.

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

В БЭВМ есть возможность создания подпрограмм, обычный аналог функции в привычных нам языках программирования. Использование подпрограмм сильно улучшает код: вы избавляетесь от дубликата кода, что как экономит итак малое адресное пространство, так и делает программу читабельнее и логичнее. Но не всегда использование подпрограмм экономит память, здесь все зависит от того, сколько раз вы вызываете подпрограмму, и сколько нужно ячеек под ее вызов.
В подпрограмму можно передавать аргументы. Самый простой, быстрый и удобный способ - через аккумулятор. Мы просто записываем в регистр общего назначения нужный нам аргумент, вызываем подпрограмму и работаем с тем же регистром. В БЭВМ для работы с подпрограммами есть 2 команды: CALL и

RET, соответственно вызов подпрограммы из любого места и выход из подпрограммы.
Но использование аккумулятора ограничивает нас в передаче аргументов, поэтому мы можем использовать стек. Он растет с вниз с ареса 7FF Это структура данных, работающая по принципу LIFO (last in first out). В БЭВМ для работы с ним используется регистр SP - указатель на вершину стека. И при вызове подпрограммы мы кладем в стек адрес возврата (текущий адрес, из которого вызвана подпрограмма).
Также через этот стек мы можем передавать аргументы в подпрограмму и получать результат работы подпрограммы при помощи использования стека. Для этого есть команды PUSH и POP, которые соответственно кладут в стек значение из AC, уменьшая значение SP на 1, и загружает значения с вершины стека, увеличивая SP на 1.
Циклы исполнения CALL и RET
Ex CALL:

DR -> BR
IP -> DR
BR -> IP
SP - 1 -> SP, AR
DR -> MEM(AR)

Ex RET:

SP -> AR
MEM(AR) -> DR
DR -> IP
SP + 1 -> SP

В доке того года хорошо прописаны следующий пункты

Позиционно-независимый код

Код, который работает относительного того адреса на который загружен.

- Необходим для, например, модулей ядра (даже при наличии виртуальной памяти!)
- Внутри программы только относительные адреса (смещения)
- Внешние ссылки только абсолютные
- В БЭВМ есть оба вида адресации!

Загрузчик и библиотеки

Загрузчик. 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

Библиотека. Набор стандартных библиотечных функций

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

– Статические связывают вызовы функций с телом функции в процессе компиляции

– Динамические — в момент загрузки

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

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

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

Инициация обмена бывает: синхронная, асинхронная, по прерыванию.

Передача данных: синхронная, асинхронная.

Завершение обмена: синхронное (сигнал завершения приходит синхронно с вызовом), асинхронное (сигнал завершения приходит через какое-то время, не синхронно).

Драйверы

Специальная программа, которая работает с каким-либо ВУ. «Знают» о принципах работы устройства, адресах регистров, поддерживаемых режимах работы. Управляются единообразным программным интерфейсом.

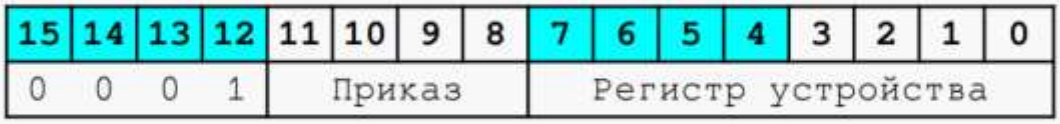
В любой операционеой системе есть API предназначеный для драйверов. В нём есть специальные входные точки, например для открытия устройства, закрытия устройства, чтения с устройства данных, записи в устройство данных, организации системы прерываний.

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

В БЭВМ реализована только программно-управляемая передача данных, т.е. только под управлением БЭВМ. Существует 10 ВУ: таймер, устройство вывода, устройство ввода, 2 устройства ввода-вывода, текстовый принтер, бегущая строка, 7-ми сегментный индикатор, клавиатура, цифровая клавиатура. Устройства предоставлены с 8-ми разрядными регистрами данных.
БЭВМ общается с контроллером внешнего устройства, через шину ВУ, а контроллер уже с самим ву через периферийную шину. В контроллере есть регистр данных, хранящий данные для ввода или вывода, регистр состояния, показывающий состояние готовности обмена данными ВУ с БЭВМ, и регистр управления, для организации прерываний. Также у каждого ВУ есть дешифратор адреса, чтобы понять, к этому ВУ идет обращение или нет.

Для работы с ВУ в БЭВМ есть следующие команды:

Команда ввода-вывода



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

Здесь наверное особых пояснений не требуется. Хочется сказать, что при вызове команд ввода-вывода, из CR поступает 8-11 биты идут на дешифратор приказа, чтобы понять, какая сейчас команда.
Если ВУ готово для обмена данными, то при считывании флага готовности с этого устройства, он запишется в 6 бит аккумулятора.
----------Булат

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

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

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

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

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

----------Максим

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

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

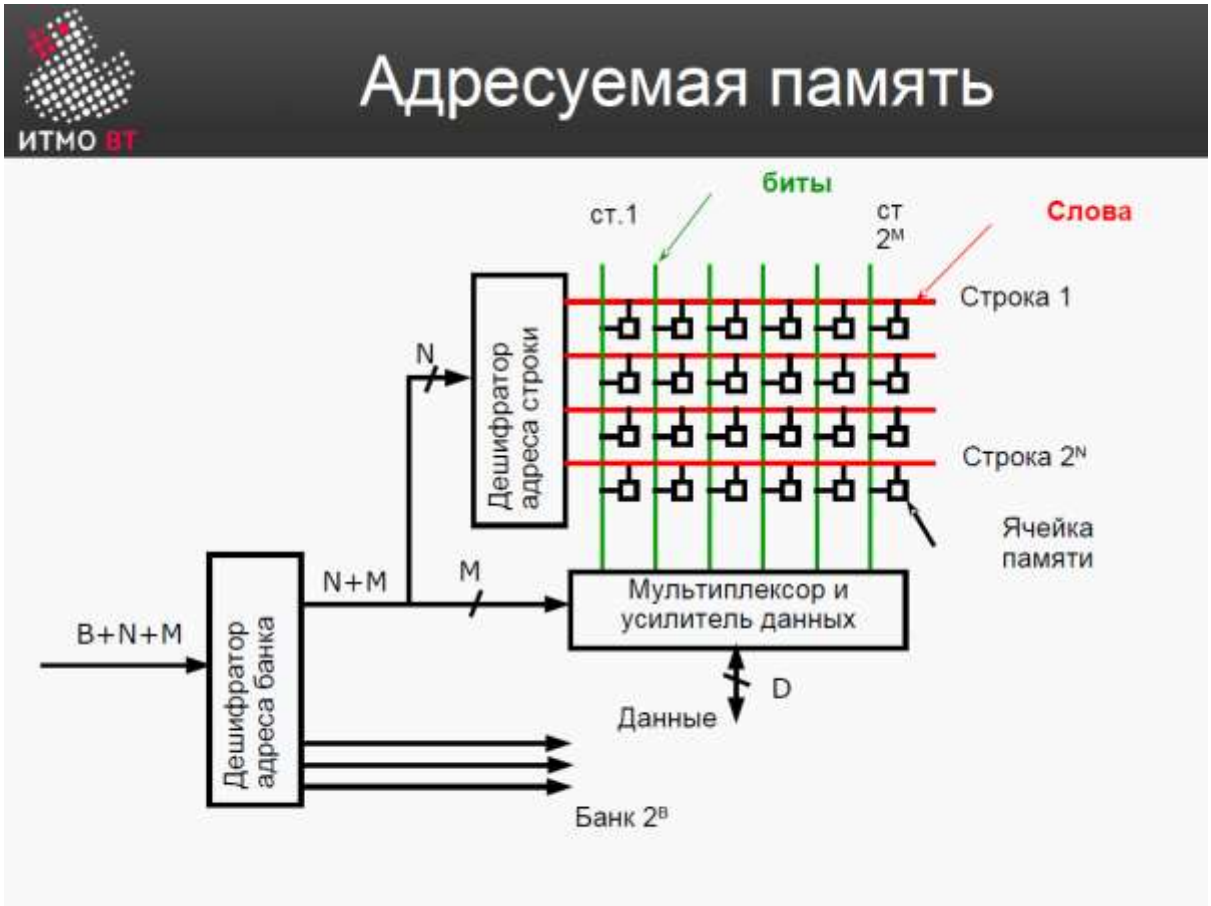
23. Архитектура ЭВМ. Гарвардская и фон-Неймановская архитектура. Организация обмена архитектуры ЭВМ с использованием шин.

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

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

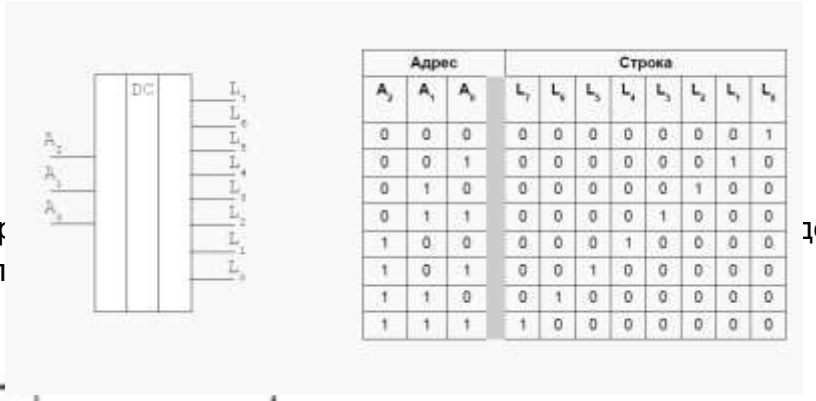
----------Боря

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



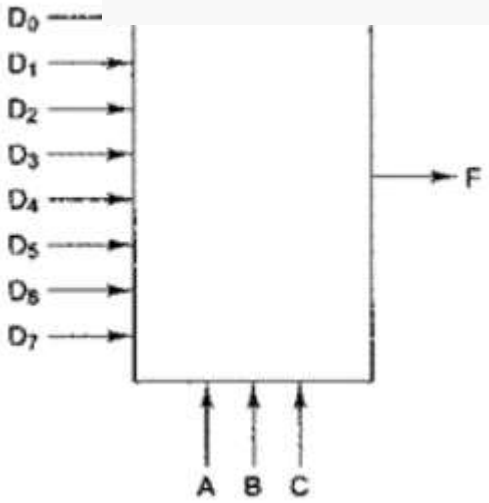
Сначала разберемся с обозначениями. B - код банка памяти. N - строка, M - столбец. Что такое банк памяти? Банк памяти - это блок динамической памяти, к которому процессор может обращаться **параллельно** с другими банками, находящимися на **той же самой микросхеме.**
Что есть дешифратор? Дешифратор - это просто логическая схема для выборки нужного провода по его двоичному номеру. Дешифратор содержит n входов и 2^n выходов. Допустим у вас есть 8 проводов на выходе дешифратора, пронумерованных от 0 до 7 и вы хотите нагрузить провод с

номером 6. Тогда вы просто нагружаете первый и второй вход дешифратора (оставляя в нуле третий). То есть получается 110, что соответствует числу 6.



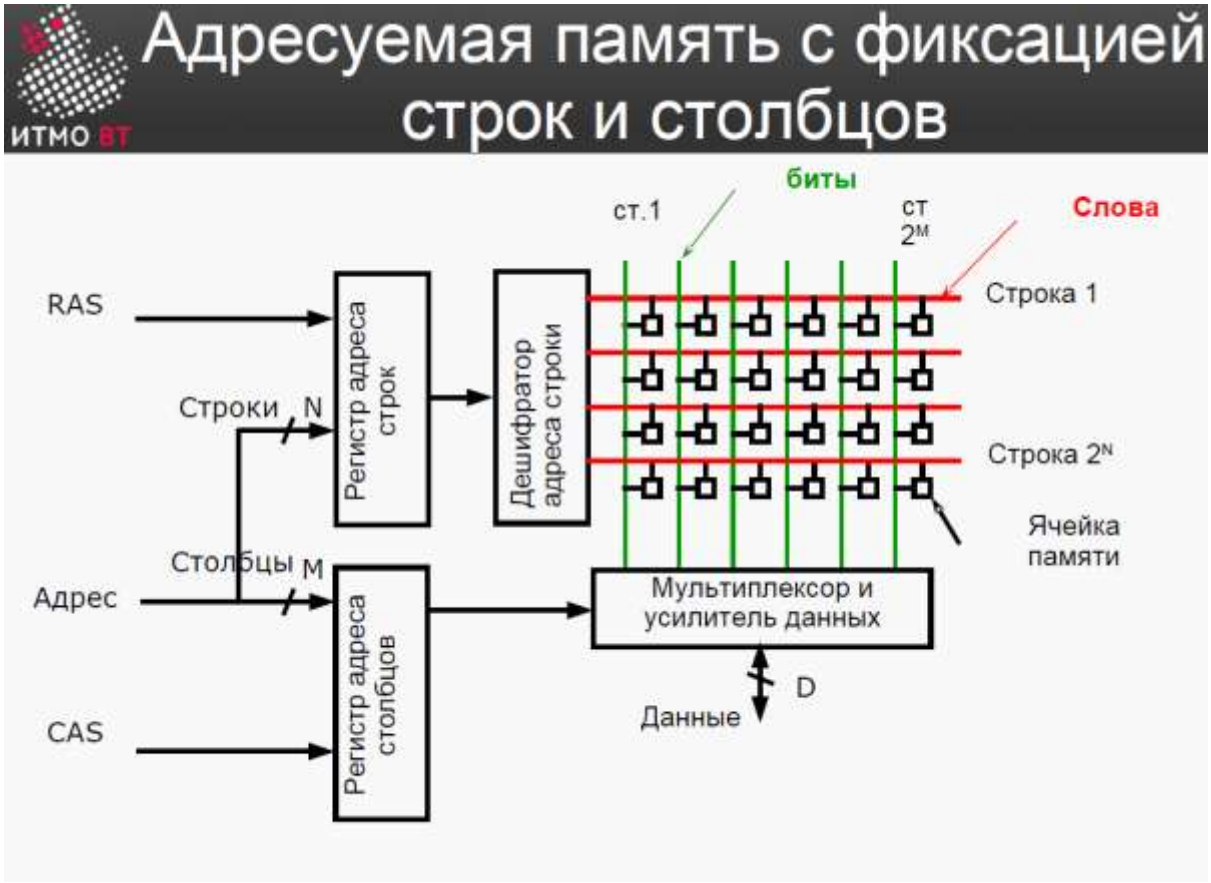
Адрес			Строка							
A_2	A_1	A_0	L_7	L_6	L_5	L_4	L_3	L_2	L_1	L_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Теперь р дешифратор, но всё же п

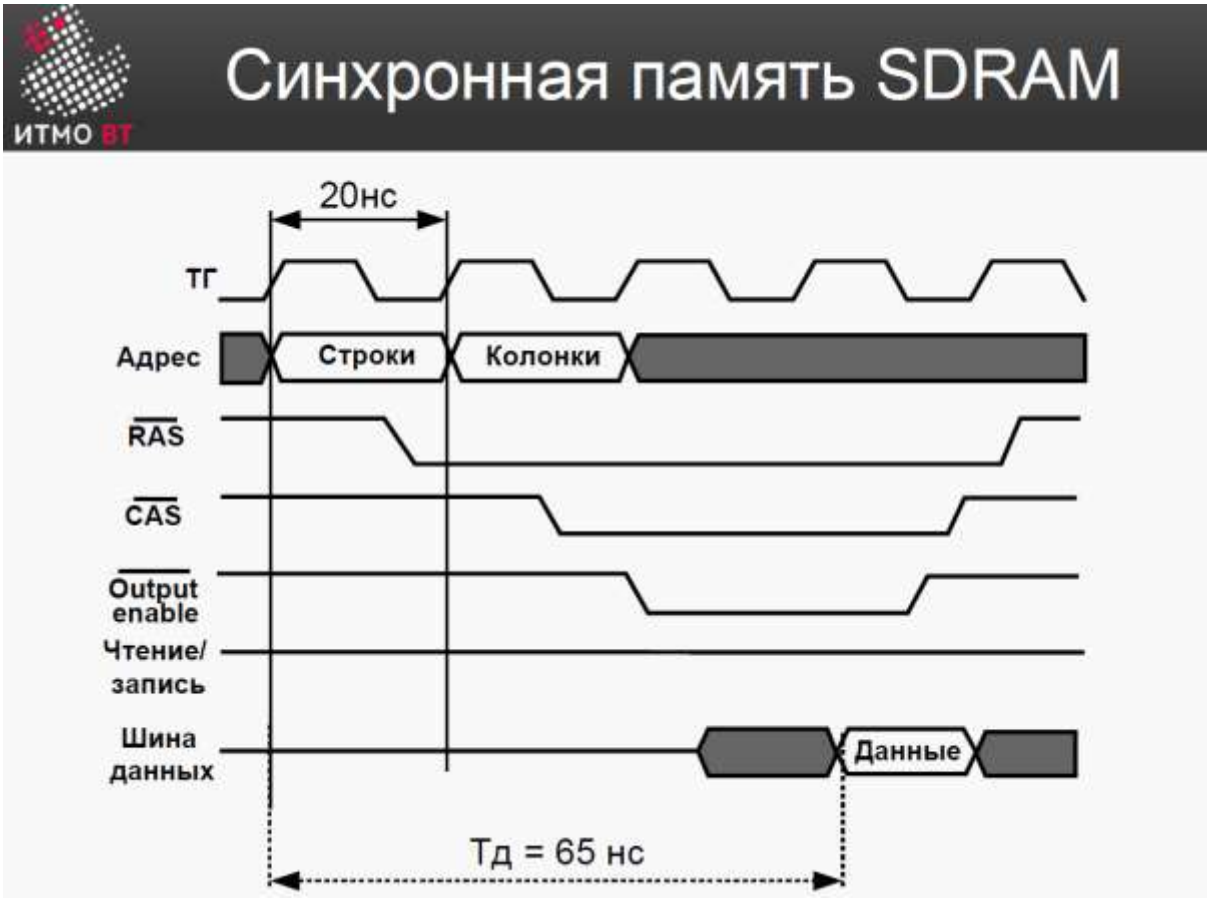


У мультиплексора есть (2^n + n) входов и всего один выход. Те n входов, что внизу, называются линиями управления. ТО ЕСТЬ у нас могут быть нагружены все 8 боковых входов, но КАКОЙ из них пойдет на ВЫХОД определяет то, что мы подадим на ЛИНИИ УПРАВЛЕНИЯ. Нах надо спросите вы? Прикол в том что это супер экономия. У НАС ВСЕГО ОДИН ВЫХОД. А теперь представьте сколько запоминающих элементов в современной памяти и подумайте сколько долларов можно сохранить используя мультиплексор.
ТЕПЕРЬ вернемся на первую схему.
По адресу банка с помощью дешифратора выбираем нужный нам банк. Далее нам нужно выбрать ячейку памяти по строке и столбцу. Строка выбирается с помощью дешифратора, а столбец (а значит и ячейка целиком, ведь строку мы уже зажгли) с помощью мультиплексора. ВСЁ!

Так, теперь разберемся с временными диаграммами. Для начала со считыванием. Допустим, мы хотим прочитать из памяти данные. Для этого первым делом обязательно на шину адреса выставляем АДРЕС (логично). Как только адрес выставлен, можно читать и мы подаем на шину управления сигнал “ЧТЕНИЕ” (на самом деле он подается одновременно с установкой адреса). Этот сигнал запускает все те процессы с выборкой ячейки памяти, о которых написано выше. Естественно, это происходит не мгновенно, и между подачей сигнала и появлении данных на шине данных пройдет какое-то время. Это время называется ВРЕМЯ ДОСТУПА (Тд). Вместе с полученными данными одновременно на шину управления поступает сигнал ГОТОВНОСТЬ, говорящий о том, что мы готовы читать далее, при этом с шины адреса будет снят адрес уже прочитанной ячейки и пропадет сигнал чтение, а вместе с ним и данные которые лежат на шине данных (ИХ нужно прочитать с шины пока они не пропали). На это тоже нужно время - время ВОССТАНОВЛЕНИЯ (Тв).
С записью почти все то же самое. Мы должны положить на шину данных ДАННЫЕ, на шину адреса АДРЕС куда записываем ДАННЫЕ и подать сигнал на запись. Этот сигнал должен дойти до линии W/R ВНУТРИ ПАМЯТИ!!! И именно время между подачей сигнала на шину управления до его доставки на линию W/R будет временем доступа. Импульс записи попав на W/R инициирует запись в память, после чего будет выставлен сигнал “Готовность”, который инициирует снятие нагрузки с шин (восстановление).



Посмотрим теперь на такую схему. Она почти такая же, что и была в самом начале. Минус первой схемы в том, что она технически сложно реализуема из-за огромного количества контактов, ведь мы целиком передавали банк и адрес строки + столбец. В данном случае другой подход: мы разделяем передачу строки и столбца на две части. Идут они всё так же по одной шине, но добавляются сигналы RAS (Row Address Strobe) и CAS (Column Address Strobe). В момент подачи сигнала RAS на регистр адреса строк, регистр прочитает адрес строки с шины. Затем на шину подается адрес столбца и вызывается сигнал CAS, помещая адрес в нужный регистр. Всё остальное - точно так же как и на первой схеме. Но справедливый вопрос Клименкову: разве мы не увеличили выборку адресов в два раза, разделив подачу адреса строки и столбца? Как бы да, но нет. В этих регистрах встроен авто-инкремент адресов, так что когда мы читаем данные последовательно расположенные в памяти, нам не нужны никакие сигналы и данные с шин, все происходит автоматически. “Люди умные”.

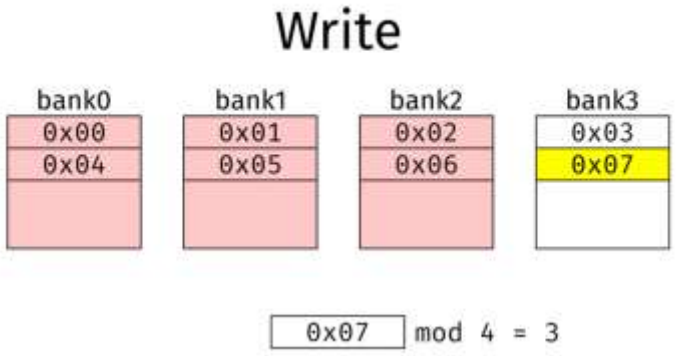


Посмотри теперь на временную диаграмму нашей SDRAM (Synchronous Dynamic Random Access Memory). Разберем сначала название. С RAM я думаю все и так ясно. Synchronous - те или иные действия будут происходить в четкий промежуток времени, определяемый ТАКТОМ. Собственно об этом нам и говорит первая строчка на схеме ТГ. Dynamic - динамическая память, в отличие от статической памяти, основанной на триггерах (данные не обновляются пока не перезапишутся), обновляет своё состояние каждый такт, но подробнее об этом лучше почитайте в интернете. Сигналы RAS и CAS в данном случае подаются инвертированные, так удобнее производителю, сути это не меняет. В один такт, собственно, врубается RAS, затем - CAS, далее сигнал Output Enable, подключающий собственно выход памяти, инициируя загрузку. Данные появляются на шине данных и всё кул, но время доступа ограничивается тактовой частотой, так что это не самая быстрая память (именно потому в кэше используют статическую память).

Теперь о **конструктивных особенностях современной памяти**:

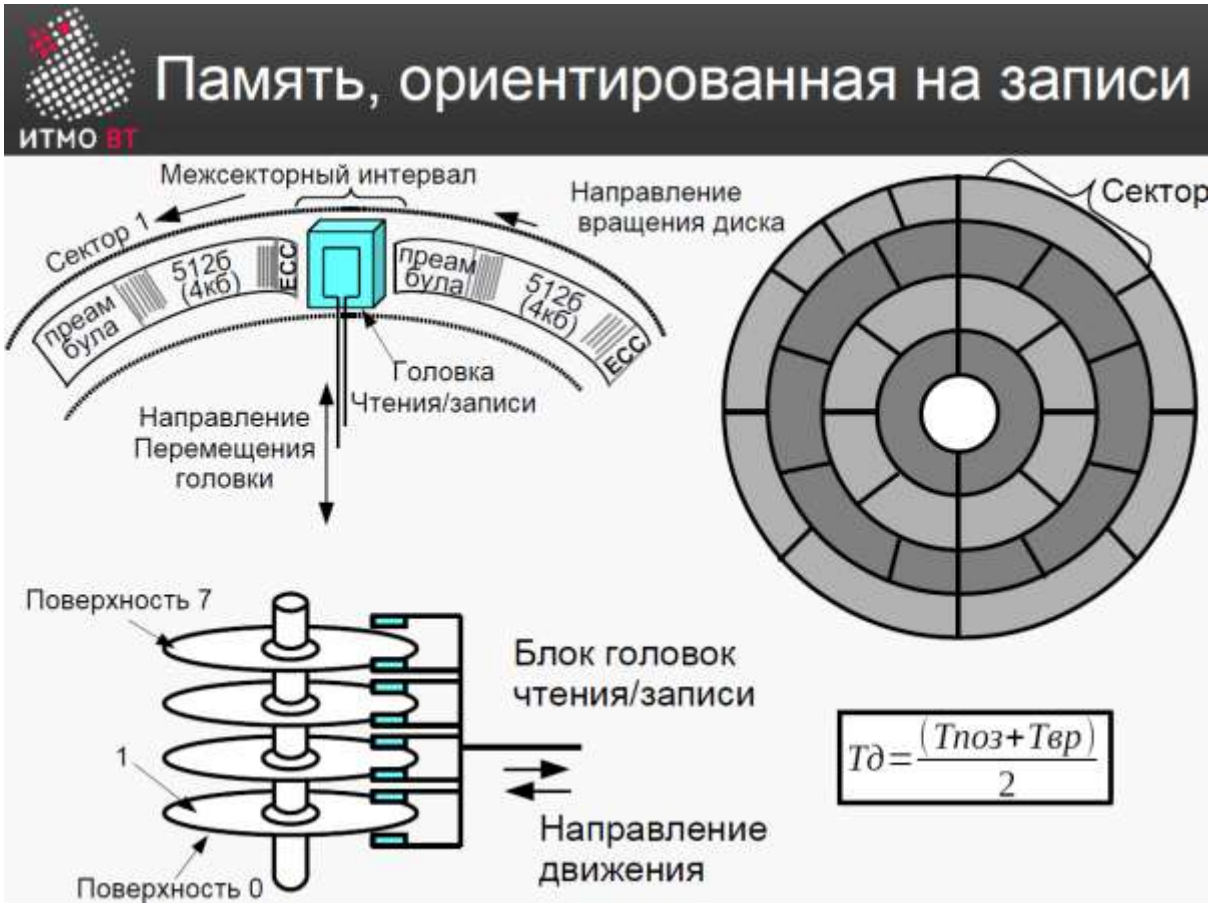
1. Burst mode - пакетный режим. То о чём говорилось парой строчек выше. Способность памяти эффективно работать с последовательными данным (отсутствие необходимости постоянно обновлять адреса при чтении последовательных данных). То есть данные передаются сразу ПАКЕТОМ. Обычно размер пакета соответствует строке кэша процессора, то есть можно быстро заполнить целиком весь кэш.

2. Double Data Rate - передача данных по фронту и по спаду. Исторически данные передаются по фронту сигнала тактового генератора, т.е в тот момент когда напряжение увеличивается (с точки зрения дискретности это переход с “0” на “1”). DDR-память в свою очередь передает данные и во время спада сигнала тактового генератора, т.е перехода с высокого напряжения на низкое.
3. SPD (Serial Presence Detect) - чип, содержащий идентификационную информацию. Информация из этого чипа (например: объем памяти, частота) считывается еще на этапе самодиагностики BIOS до старта операционной системы. Исходя из этой информации компьютер может сам установить оптимальные параметры для работы с конкретной памятью.
4. Interleaving - расслоение памяти, повышает производительность. Тут чуть посложнее. Помним, что наша память состоит из **банков** - независимых друг от друга частей памяти на одной схеме, к которым можно осуществлять параллельный доступ. Interleaving использует идею расположения набора данных не в одном банке, а сразу в нескольких. То есть мы кладем данные не друг за другом в одном банке, а параллельно в нескольких. Таким образом, параллельно обращаясь к разным банкам памяти мы увеличиваем скорость доступа. Требуется поддержка материнской платы.

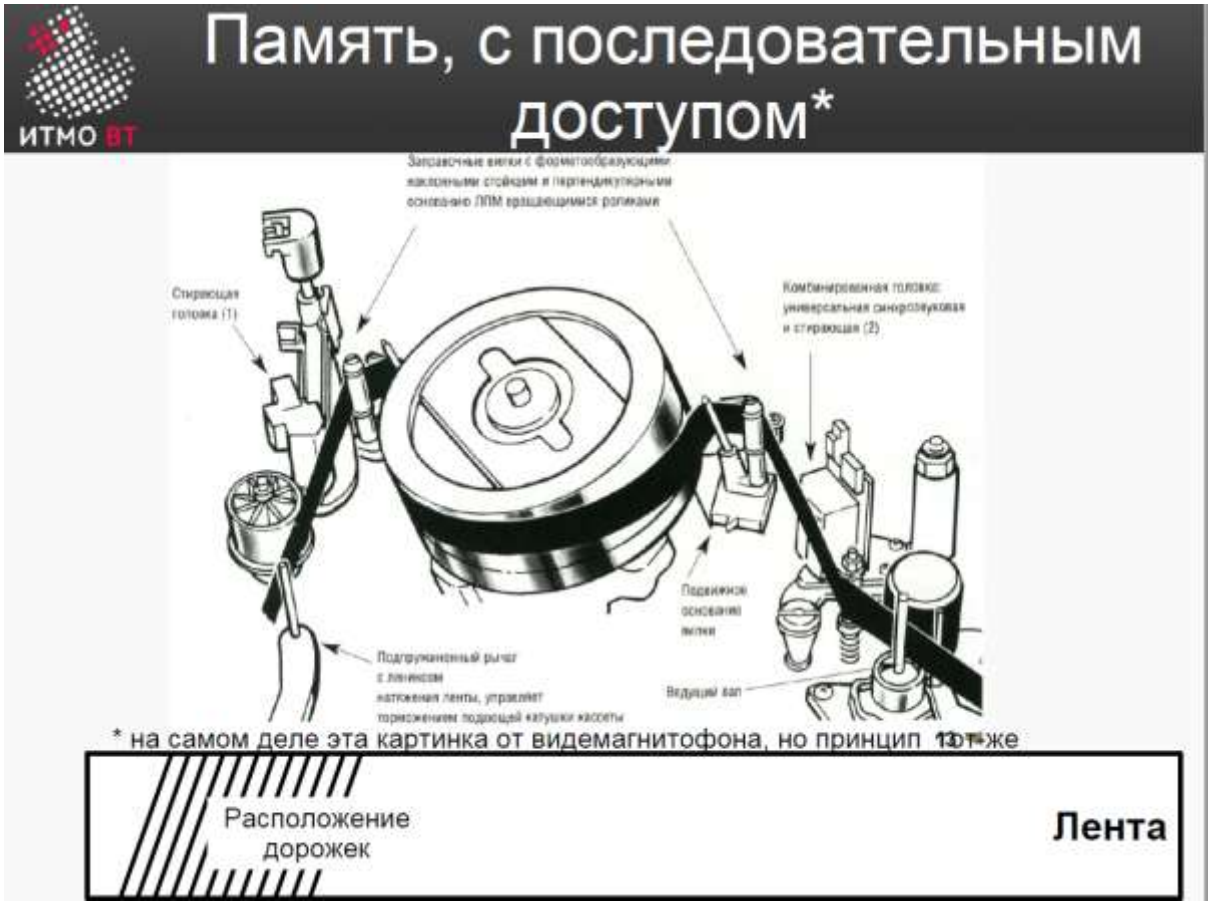


5. DDR4-2133 8192 MB **PC4-17000.** Разберем. DDR4 - тип памяти, 2133 - частота в МГц. Далее следует объем. Последняя штука, согласно Клименкову, это индекс производительности относительно какой-то эталонной величины. Однако гугл говорит, что это просто количество передаваемых мегабайт в секунду, которое можно получить умножив на 8 частоту памяти (Per Clock Rate), ибо за один такт мы передаем 8 байт данных.

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

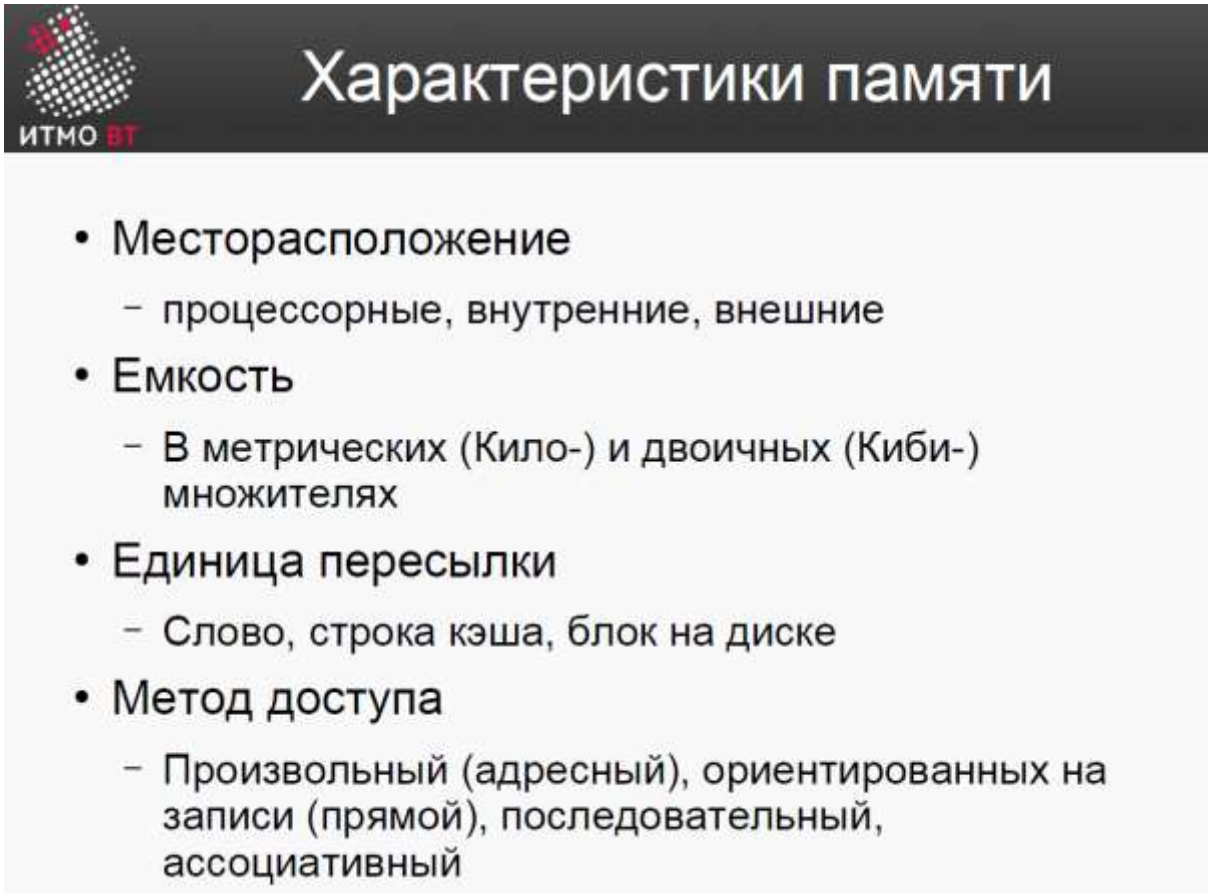


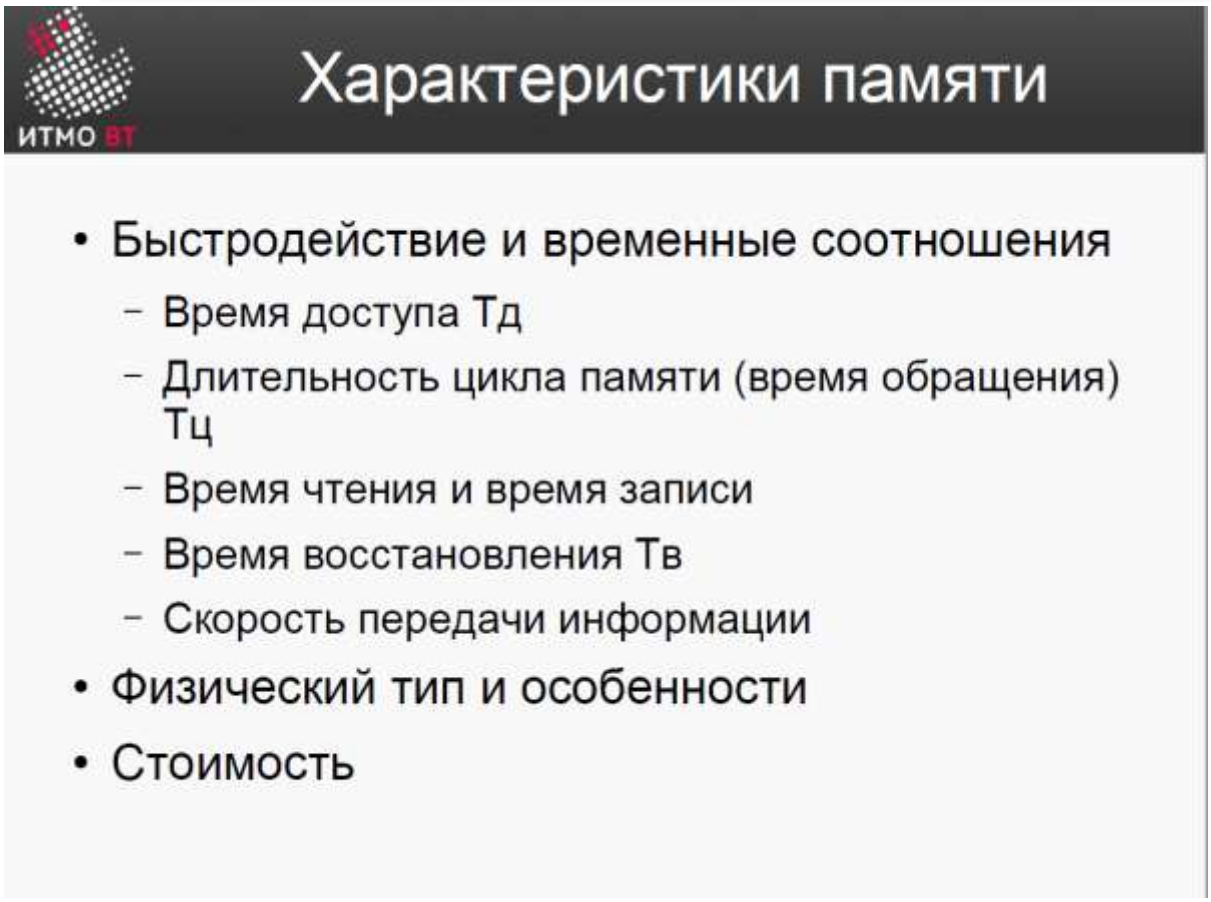
Об устройстве жесткого диска. Жесткий диск состоит из нескольких ферромагнитных поверхностей, на которые “намагничиваются” данные. Каждая поверхность разбита на концентрические кольцевые области - дорожки. Каждая дорожка разбита на равные отрезки - секторы, являющиеся минимально адресуемой частью памяти жесткого диска. Сектор содержит преамбулу - данные, позволяющие головке жесткого диска синхронизироваться перед чтением или записью. Затем следует сами данные (обычно 512 байт, но на оптических дисках может быть до 4 КБайт). В конце следует ECC (Error Correcting Code), позволяющий обнаружить и исправить ошибки.
Считыванием и записью данных занимается головка, которая перемещается от центра к краю, выбирая нужную дорожку. После того, как дорожка выбрана, головка должна дождаться, пока вращающаяся поверхность установит нужный сектор под головкой, тогда и происходит чтение/запись. Время доступа в таком случае считается как время полувращения + время полупозиционирования.Стоит отметить, что данные записанные ближе к краю будут читаться быстрее, чем те, которые ближе к центру, так как линейная скорость вращения блина относительно головки прямо пропорциональна радиусу.



Здесь речь идет о ленточных накопителях. Тут всё просто: ленточные накопители ужасно медленные, но цена за единицу объема памяти крайне мала, поэтому на них хранят огромные массивы данных, которые нужны, например, на случай, если потребуется бэкап.
Скорость доступа определяется скоростью прокрутки этой самой ленты, но механика не позволяет сделать эту скорость очень высокой без потери надежности. Поэтому современные ленточные накопители работают по принципу наклонно-строчной магнитной записи (как показано на слайде). Лента проходит под вращающейся головкой под некоторым углом, обеспечивая бОльшую плотность записи и скорость (однако на данный момент уже придумали способ достичь такой скорости обычным линейным методом)

28. Характеристики запоминающих устройств. Пирамида памяти.

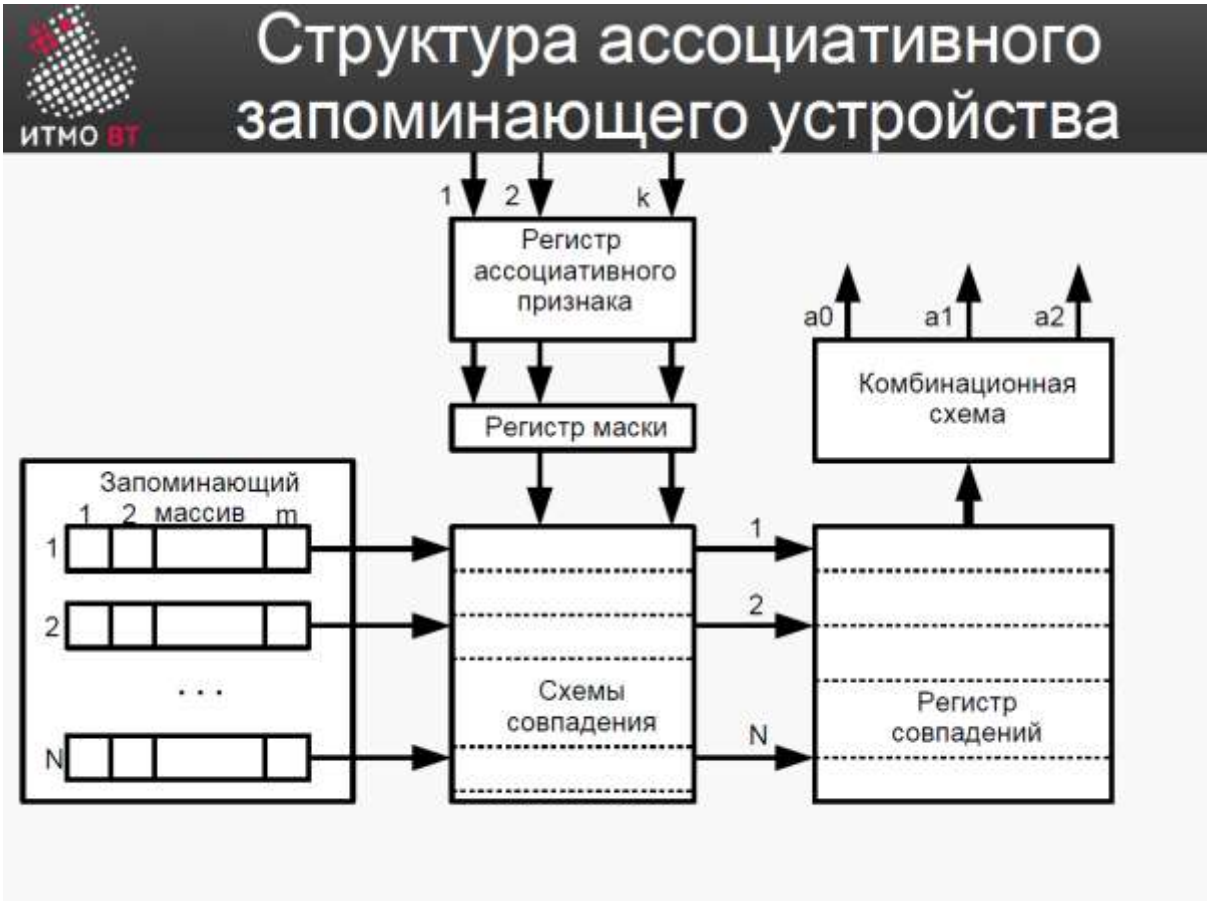




(вроде как все эти вещи были рассмотрены выше)

Это то, что следует просто зазубрить. В данной таблице показаны основные характеристики различных типов памяти. Самая быстрая, но самая маленькая по объему память - это регистры процессора. Очень хорошо, если ваша программа будет оперировать только с ними, но в мощных современных программах это невозможно. Далее идёт кэш первого-третьего уровней, как видим, памяти стало больше в несколько десятков или сотен раз, но при этом и существенно упало время доступа. Далее принцип такой же: чем объемнее ваша память, тем она медленнее. Быстрая память очень дорогая и её очень мало. В столбце помеченном “*” показана относительная разница во времени доступа, где за эталон взят регистр процессора. Видим, что ОЗУ медленнее регистров процессора в 50-300 раз. Это стоит учитывать при проектировании программ.

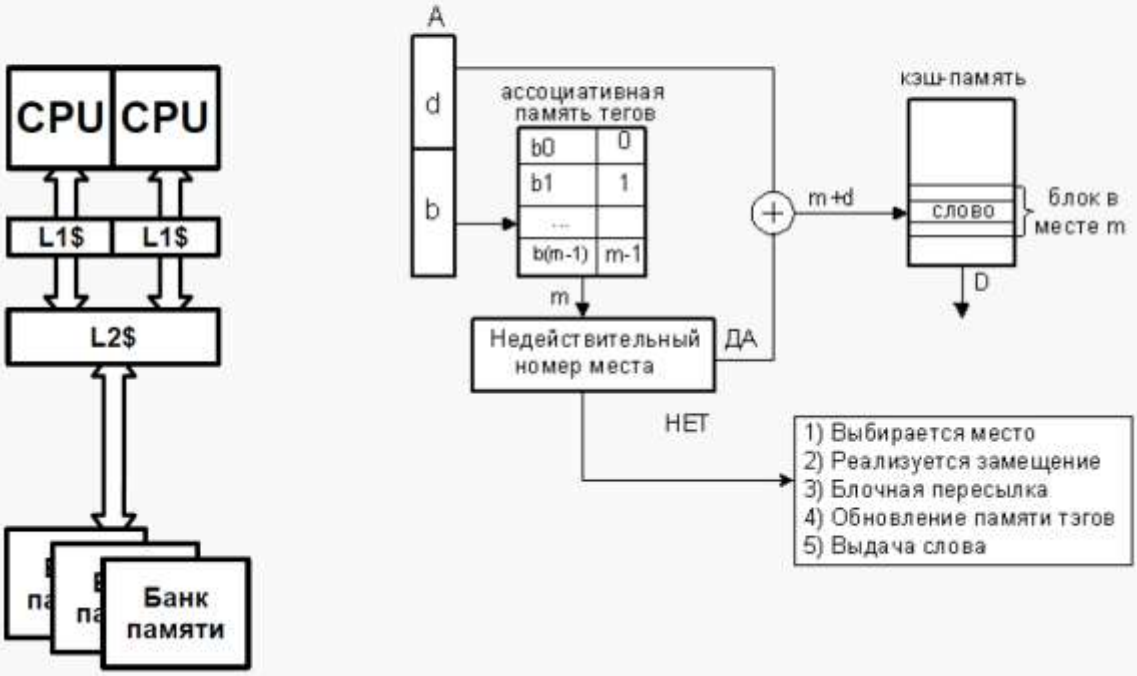
29. Ассоциативная память, Кэш-память. Влияние промахов кэш-памяти на производительность.



Ассоциативная память. Память, рассмотренная выше, адресовалась по номеру столбца и строки. Мы доставали нужные нам данные по их местоположению. В ассоциативной памяти подход другой: мы достаем нужные данные по их содержанию, а не их местоположению. Для этого каждая ячейка имеет в себе компаратор, которые может обнаружить соответствие. Найденная ячейка попадает в регистр совпадений и далее на комбинационную схему. То есть это как бы поиск по ключу, которому в соответствие ставятся те или иные структуры данных. Такая память гораздо быстрее чем обычная RAM, но и стоит дороже за счёт компараторов и более трудной логики. Принцип ассоциативной памяти использует кэш.

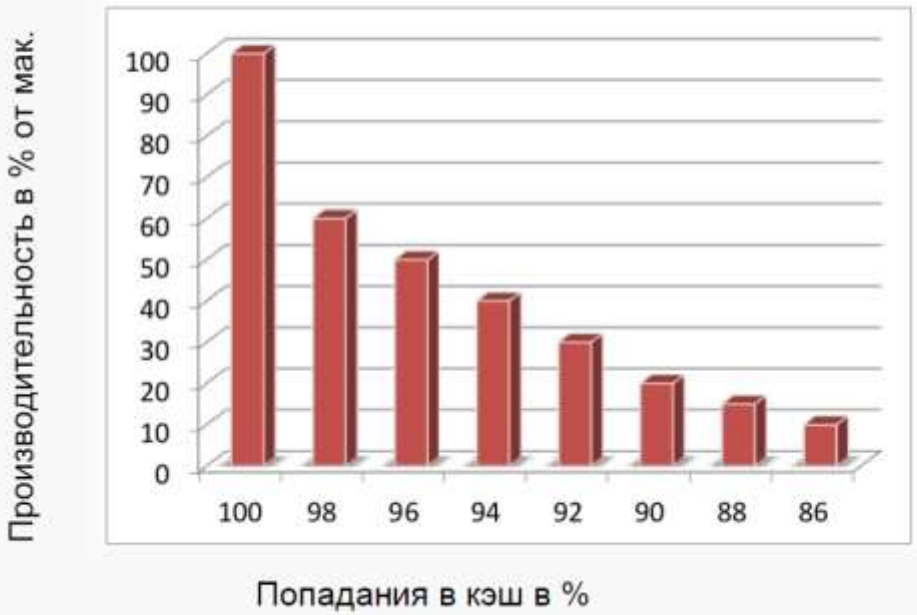
ИТМО ВТ

Кэш память



ИТМО ВТ

Влияние промахов кэш-памяти



На данной схеме показано, как кэш-промахи влияют на скорость обработки данных. Как было показано на пирамиде памяти, RAM медленнее кэша на 1-2 порядка. Отсюда, если мы неоптимально будем использовать кэш, производительность будет падать многократно. Согласно диаграмме 2% промаха забирают 40% производительности. 15% промахов забирают >90% производительности.

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

----------Толя

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

Понятие сети ЭВМ - это совокупность средств вычислительной техники с средствами телекоммуникации.

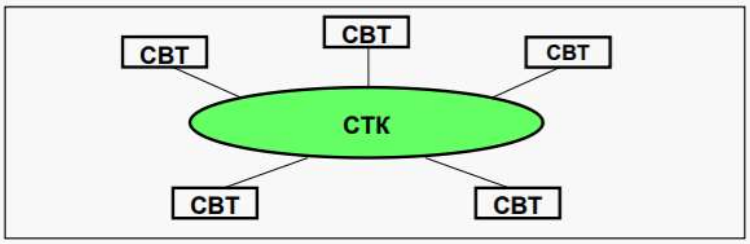
Средства вычислительной техники - это могут быть отдельные ЭВМ, вычислительные комплексы и вычислительные системы.

Вычислительный комплекс - это совокупность ЭВМ, которые объединены определенной задачей.

Вычислительные системы - это вычислительный комплекс + прикладное ПО, которое мы установили.

ИТМО ВТ

Понятие сети ЭВМ



- Средства вычислительной техники (СВТ): ЭВМ, вычислительные комплексы (ВК) и вычислительные системы (ВС) - реализуют обработку данных.
- Средства телекоммуникаций (связи) (СТК): совокупность каналов связи и каналообразующей аппаратуры - реализуют передачу данных.
- Сеть ЭВМ (вычислительная сеть, компьютерная сеть) = СВТ+СТК

Классификация компьютерных сетей:

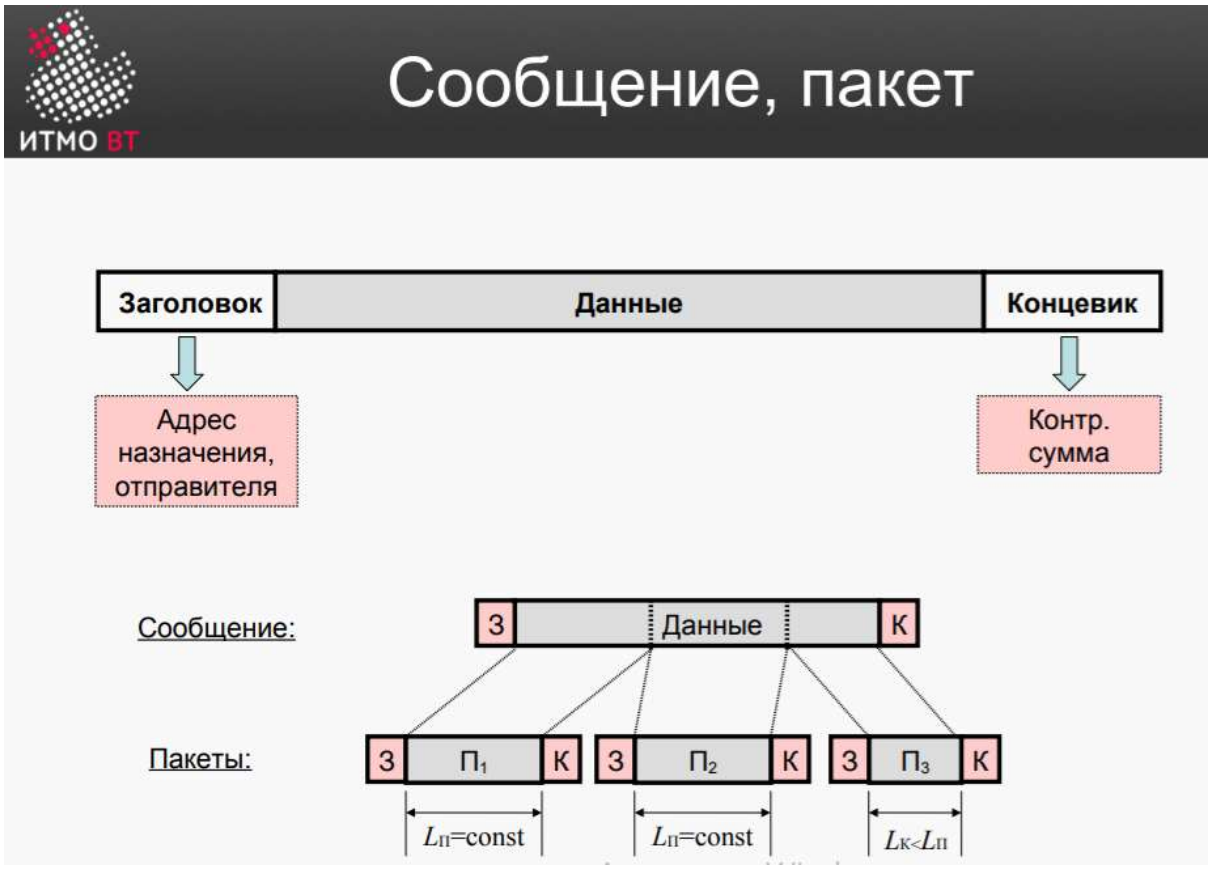
1. Сети можно классифицировать по размеру:
    a. PAN(personal area network) - сеть, объединяющая персональные электронные устройства(bluetooth)
    b. LAN(local area network) - локальная сеть
    c. MAN(metropolitan area network) - Сети города. Объединяет в себе технологии LAN и WAN.
    d. WAN(wide area network) - глобальная сеть
2. По принадлежности:
    a. Офисные
    b. Корпоративные
    c. Частные VPN
3. По назначению:
    a. Вычислительные - используется для счетных задач.
    b. Информационные - для передачи информации(например, видео).
    c. Информационно-вычислительные - гибрид первых двух.
    d. Информационно-управляющие - в основном нужны для задач управления.
4. По области применения:
    a. SAN - сети хранения данных.
    b. Серверные фермы.

5. Прочие:
    a. Иерархические.
    b. Беспроводные.
    c. Виртуальные VLAN.

Сообщение и пакет:

Данные разбиваются на пакеты, которые последовательно передаются. Это сделано для того, чтобы один процесс передачи данных не занимал слишком много времени.

Концевик - хэш(раньше этот хэш собирался так: мы преобразуем наши данные в байты и складываем их). Нужен он для проверки корректности передачи данных.



Модель взаимодействия открытых систем(OSI):

- Состоит из нескольких уровней(они перечислены на картинке ниже).
- Наше сообщение бьётся на пакеты и на каждом уровне они оборачивается в заголовок уровня(чтобы понять, что это значит, почитайте следующие билеты).

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

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

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

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

----------Кирилл

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

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

38. Контроллер передачи асинхронного последовательные интерфейса.

39. Контроллер приема асинхронного последовательные интерфейса.

40. Организация прямого доступа к памяти. Контроллер ПДП.

маме привет

Отошел, оставил курсор здесь: