

Выдержки для подготовки к экзамену по ОПД

Авторы

[Ондрей](#)

Макс Антонов

Вовочка

Юра Бобален

ЕгорВышеГор

Tarasenko VLadd

Пиво Ильинское

Выдержки для подготовки к экзамену по ОПД

1

1. Две формы представления информации. Способы представления дискретной информации. Системы счисления, используемые в вычислительной технике: двоичная, 8-я, 10-я, 16-я, двоично-десятичная.	4
2. Представление чисел с фиксированной точкой. Прямой, обратный и дополнительный код. Формирование битовых признаков переноса, переполнения, отрицательного результата, нуля.	5
3. Представление символьных и строковых данных. Принципы построения кодовых таблиц ASCII, КОИ-8, ISO8859-5, Windows-1251, UTF-8, UTF-16.	6
4. Базовые элементы вычислительной техники: ячейки, регистры, шины, вентили, тактовые генераторы, логические схемы, триггеры, регистры, счетчики, сумматоры.	8
5. Структура и принцип функционирования ЭВМ. Порядок функционирования простого процессора на примере калькулятора.	11
6. Операционная система Unix — ядро ОС и файловая система	13
7. Операционная система Unix — интерпретаторы, стандартные потоки ввода вывода, фильтры	14
8. Операционная система Unix — основные команды, права файлов и способы их задания	15
9. Состав и структура БЭВМ. Адресные пространства БЭВМ. Система команд БЭВМ, форматы команд. Машинные циклы	17
10. Организация вычислений в БЭВМ. Сдвиги, арифметические и логические операции. Цикл выборки команды	19
11. Организация массивов данных. Режимы адресации. Цикл выборки адреса и операнда БЭВМ	20
12. Управление вычислительным процессом в БЭВМ. Команды ветвлений, цикл исполнения команды LOOP	21
13. Подпрограммы в БЭВМ. Цикл исполнения команд перехода и возврата из подпрограммы. Стек, передача параметров. Позиционно-независимый код. Загрузчик и библиотеки.	23
14. Организация ввода-вывода в вычислительных системах. Инициация обмена, передача информации и завершение обмена. Драйверы.	26
15. Организация ввода-вывода в БЭВМ. Устройства ввода-вывода, команды.	29
16. Организация асинхронного обмена в БЭВМ. Пример программы. Временные издержки асинхронного обмена	32
17. Организация прерываний в БЭВМ. Вектора прерываний, контроллер прерывания	33
18. Организация обмена по прерыванию программы в БЭВМ. Пример программы. Цикл прерывания	37
19. Понятие многоуровневой ЭВМ. Понятие и пример программы на разных уровнях	42
20. Микропрограммный уровень БЭВМ. Структура МПУ. Форматы микрокоманд	43
21. Структура и принципы работы арифметико-логического устройства и коммутатора. Регистр состояния БЭВМ	45
22. Микропрограммное управление вентильными схемами. Схема управления. Интерпретатор БЭВМ.	48
23. Архитектура ЭВМ. Гарвардская и фон-Неймановская архитектура. Организация обмена архитектуры ЭВМ с использованием шин	53
24. Архитектура многопроцессорных ЭВМ. Системный коммутатор. Архитектуры UMA и NUMA	54
25. Структура современных процессоров. Окружение процессора. CISC, RISC, VLIW	59
26. Адресуемая память, организация и временные диаграммы. Конструктивные особенности современной памяти.	65

27. Память, ориентированная на записи (блочная память). Организация дисковой памяти и памяти на магнитных лентах.	69
28. Характеристики запоминающих устройств. Пирамида памяти	72
29. Ассоциативная память, Кэш-память. Влияние промахов кэш-памяти на производительность.	74
30. Предназначение и организация виртуальной памяти. Сегментно-страничная организация. Устройство управления памятью (MMU), буфер трансляции (TLB).	77
31. Сетевые технологии, Понятие сети ЭВМ, классификация компьютерных сетей. Сообщение и пакет. Модель взаимодействия открытых систем.	81
32. Модель TCP/IP: передающая среда, канальный и сетевой уровень. Адресация, передача и маршрутизация пакетов.	Error! Bookmark not defined.
33. Модель TCP/IP: выделение адресов (DHCP), сервисы имен, транспортный и прикладной уровни.	91
34. Интерфейсы ввода-вывода. Контроллеры внешних устройств. Уровни стандартизации, сопряжения с системной шиной, циклы обмена. Регистры контроллера.	92
35. Параллельная передача данных. Контроллеры параллельной передачи и приема.	95
36. Синхронные последовательные интерфейсы. Контроллеры последовательной передачи и приема.	97
37. Асинхронный обмен. Принципы деления частоты, формат кадра	99
38. Контроллер передачи асинхронного последовательного интерфейса.	101
39. Контроллер приема асинхронного последовательного интерфейса.	103
40. Организация прямого доступа к памяти. Контроллер ПДП.	Error! Bookmark not defined.

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

Первая форма представления информации называется аналоговой или непрерывной (с помощью сходной величины – аналога). Количество значений, которые может принимать величина, представленная в такой форме бесконечно велико, даже если величина изменяется в ограниченном диапазоне. Отсюда названия – непрерывная величина и непрерывная информация. Слово непрерывность выделяет основное свойство таких величин – отсутствие разрывов, промежутков между значениями, которые может принимать аналоговая величина. Величина представляется в виде одного сигнала, пропорционального этой величине. Эта форма представления используется в аналоговых вычислительных машинах.

Вторая форма представления информации называется цифровой или дискретной (с помощью набора напряжений, каждое из которых соответствует одной из цифр представляющей 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). Примером аналогового представления графической информации может служить, например, живописное полотно, цвет которого изменяется непрерывно, а дискретного – изображение, напечатанное с помощью струйного принтера и состоящее из отдельных точек разного цвета. Примером аналогового хранения звуковой информации является виниловая пластинка (звуковая дорожка изменяет свою форму непрерывно), а дискретного – аудиокомпакт-диск (звуковая дорожка которого содержит участки с различной отражающей способностью).

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

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 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. вместо привычной десятичной системы счисления была взята двоичная. 2СС также является позиционной СС. Существуют специальные термины, широко используемые в вычислительной технике: бит, байт и слово. 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. 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 индикаторы с десятичным представлением полученных результатов. В 2-10СС десятичные цифры от 0 до 9 представляют 4- разрядными двоичными комбинациями от 0000 до 1001. Две двоично-десятичные цифры составляют 1 байт (можно представлять значения от 0 до 99)

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

Целые двоичные числа без знака можно использовать для представления нуля и целых положительных чисел. При размещении таких чисел в одном 16-разрядном слове они могут изменяться от $(0000\ 0000\ 0000\ 0000)_2 = (000\ 0)_16 = 0$ до $(1111\ 1111\ 1111\ 1111)_2 = (\text{FFFF})_{16} = 2^{16} - 1 = 65535$. Такая запись называется прямым кодом числа. Подобные числа (так же как и рассмотренные ниже двоичные числа со знаком) относятся к числам с фиксированной запятой, разделяющей целую и дробную части числа. В числах, используемых в базовой ЭВМ, положение запятой строго фиксировано после младшего бита слова. Целые двоичные числа со знаком используются тогда, когда необходимо различать положительные и отрицательные числа. В современных ЭВМ для представления целых чисел со знаком используется дополнительный код, в котором старший бит формата определяет знак числа: 0 - для положительных чисел и 1 - для отрицательных чисел. При этом дополнительный код положительного числа совпадает с его прямым кодом.

- А для представления отрицательного числа в дополнительном коде производится инвертирование прямого кода модуля числа (получение обратного кода числа) и добавление к результату единицы. Такая же операция используется при изменении знака числа, представленного в дополнительном коде. Использование дополнительного кода упрощает конструкцию ЭВМ, так как при сложении двух таких чисел, имеющих разные знаки, не требуется переходить к операциям вычитания меньшего (по модулю) числа из большего и присвоения результату знака большего числа.

Кроме того, одной и той же схемой сумматора можно воспользоваться для выполнения операций над знаковым и беззнаковым представлением числа. Признаком выхода за границы разрядной сетки для беззнакового представления числа является перенос в старший разряд (бит С - Carry). Например, при сложении:

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

В ответе должно получиться $32768+32768=65536$, но т.к. разрядность слова составляет лишь 16 бит, то в нем сохраняется только часть результата, т.е. 0. Единица, возникшая вследствие переноса оказалась в несуществующем 17 разряде. Признаком переполнения разрядной сетки для знакового представления является бит переполнения (Overflow). Разные знаки слагаемых, или совпадение знаков слагаемых со знаком суммы свидетельствуют о том что результат корректен. в противном случае формируется сигнал – Переполнение Признак отрицательного результата N при знаковом представлении выставляется в случае когда в старшем разряде числа в доп. коде находится 1 Признак нулевого результата Z выставляется в случае когда все разряды числа равны 0

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

ASCII		
Порядковые номера	Коды символов	Что за...
0 – 31	00000000 – 00011111	Управляющие символы для управления процессом вывода текста на экран или печать, подача звукового сигнала, разметка текста и т.п.
32 – 127	00100000 – 01111111	Сюда входят строчные и прописные буквы латинского алфавита, десятичные цифры, знаки препинания, всевозможные скобки, коммерческие и другие символы. (Символ 32 – пробел)
128 – 255	10000000 – 11111111	Кодовая страница для нелатинского алфавита

КОИ-8

— восьмибитовая кодовая страница, совместимая с ASCII Разработчики КОИ-8 поместили символы русского алфавита в верхней части кодовой таблицы таким образом, что позиции символов кириллицы соответствуют их фонетическим аналогам в английском алфавите из нижней части таблицы. Это означает, что если в тексте, написанном в КОИ-8, убрать восьмой бит каждого символа, то получится «читаемый» текст, подобный транслиту. Например, слова «Русский Текст» превратятся в «rUSSKIJ tEKST». Из-за этого символы кириллицы расположены не в алфавитном порядке.

ISO – 8859-5

- 8-битная кодовая страница из семейства кодовых страниц стандарта ISO-8859 для представления кириллицы. Нижняя часть таблицы кодировки полностью соответствует кодировке ASCII(за такое можно и бан словить, давно ASCII стала 8ми битной? совпадает начало, потому что ASCII 7 ми битная). Основной недостаток - отсутствие некоторых символов, такие как тире (—), кавычки-ёлочки («»), градус (°), поэтому в России почти не использовалась.

Windows-1251

— набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии. Windows-1251 как и KOI8-R выгодно отличается от других 8-битных кириллических кодировок (таких как CP866 и ISO 8859-5) наличием практически всех символов, использующихся в русской типографике для обычного текста
Минусы:

- Строчная буква «я» имеет код 0xFF (255 в десятичной системе), что совпадает со служебным символом в некоторых других кодировках, например в CP437 это «неразрывный пробел»
- Отсутствуют символы псевдографики, имеющиеся в CP866 и KOI8

UTF-8

— одна из общепринятых 8-битных и стандартизованных кодировок текста, которая позволяет хранить символы Юникода, используя переменное количество байт (от 1 до 6) Совместимость с ASCII — любые их 7-битные символы отображаются как есть, а остальные выдают пользователю мусор (шум). Поэтому в случае, если латинские буквы и простейшие знаки препинания (включая пробел) занимают существенный объём текста, UTF-8 даёт выигрыш по объёму по сравнению с UTF-16.

Кодирование UTF-8 1.

Если размер символа в кодировке UTF-8 = байт Код имеет вид (0aaa aaaa), где «0» — просто ноль, остальные биты «а» — это код символа в кодировке ASCII; 2. Если размер символа в кодировке в UTF-8 байт (то есть от до):

2.1 Первый байт содержит количество байт символа, закодированное в единичной системе счисления;

2 — 11

3 — 111

4 — 1111

5 — 1111 1

6 — 1111 11

2.2 «0» — бит терминатор, означающий завершение кода размера

2.3 далее идут значащие байты кода, которые имеют вид (10xx xxxx), где «10» — биты признака продолжения, а «х» — значащие биты.

В общем случае варианты представления одного символа в кодировке UTF-8 выглядят так:

(1 байт) 0aaa aaaa

(2 байта) 110x xxxx 10xx xxxx

(3 байта) 1110 xxxx 10xx xxxx 10xx xxxx

(4 байта) 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx

(5 байт) 1111 10xx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx

(6 байт) 1111 110x 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx

UTF-16

— символы кодируются двухбайтовыми словами с использованием всех возможных диапазонов значений (от 0000_{16} до $FFFF_{16}$). При этом можно кодировать символы Unicode Сдвигаем символ в 8в диапазонах $0000_{16}..D7FF_{16}$ и $E000_{16}..10FFFF_{16}$. Исключенный отсюда диапазон

6

$D800_{16}..DFFF_{16}$ используется как раз для кодирования так называемых суррогатных пар — символов, которые кодируются двумя 16-битными словами. Символы Unicode до $FFFF_{16}$ включительно (исключая диапазон для суррогатов) записываются как есть 16-битным словом. Символы же в диапазоне $10000_{16}..10FFFF_{16}$ (больше 16 бит) уже кодируются парой 16-битных слов. Для этого их код арифметически сдвигается до нуля (из него вычитается минимальное число 10000_{16}). В результате получится значение от нуля до $FFFF_{16}$, которое занимает до 20 бит. Старшие 10 бит этого значения идут в лидирующее (первое) слово, а младшие 10 бит — в последующее (второе). При этом в обоих словах старшие 6 бит используются для обозначения суррогата. Биты с 11 по 15 имеют значения 11011_2 , а 10-й бит содержит 0 у лидирующего слова и 1 — у последующего.

7

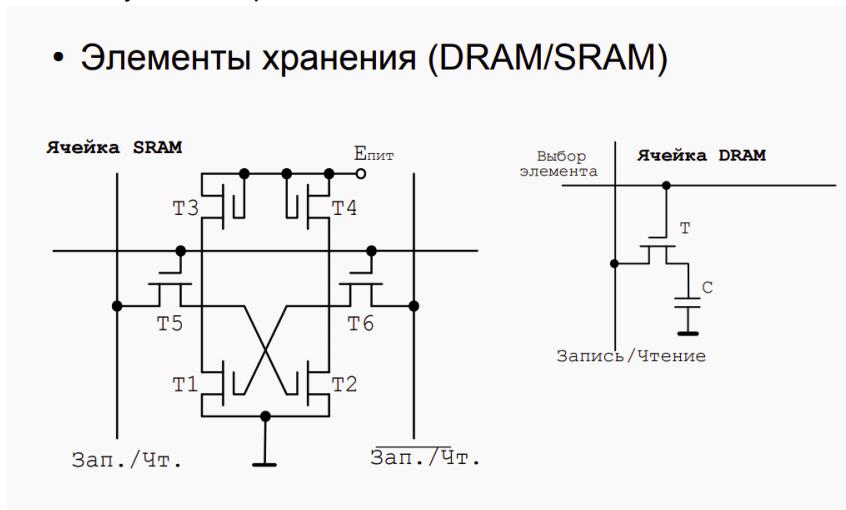
4. Базовые элементы вычислительной техники: ячейки, регистры, шины, вентили, тактовые генераторы, логические схемы, триггеры, регистры, счетчики, сумматоры.

Ячейка памяти – минимальный адресуемый элемент запоминающего устройства ЭВМ. Ячейки имеют адрес (порядковый номер, число), по которому к ним могут обращаться команды процессора. Ячейки памяти состоят из элементов, которые могут находиться в одном из двух устойчивых состояний: конденсатор заряжен или разряжен, транзистор находится в проводящем или непроводящем состоянии. Одно из таких физических состояний создает высокий уровень выходного напряжения элемента памяти, а другое – низкий. Первое обычно принимается за двоичную 1, а второе – за двоичный 0. Возможно и обратное кодирование. Память бывает статическая (SRAM - static random access memory) и динамическая (DRAM – dynamic ...)

Статическая память с произвольным доступом (SRAM, static random access memory) — полупроводниковая оперативная память, в которой каждый двоичный разряд хранится в схеме с положительной обратной связью, позволяющей поддерживать состояние без регенерации. Тем не менее, сохранять данные без перезаписи SRAM может, только пока есть питание. Используется в кэшах

Динамическая память с произвольным доступом — тип компьютерной памяти, отличающийся использованием полупроводниковых материалов, энергозависимостью и возможностью доступа к данным, хранящимся в произвольных ячейках памяти. Физически DRAM состоит из ячеек, созданных в полупроводниковом материале в виде емкости. Заряженная или разряженная емкость хранит бит данных. Каждая ячейка такой памяти имеет свойство разряжаться, поэтому их постоянно надо подзаряжать. Совокупность ячеек образует условный «прямоугольник», состоящий из определенного количества строк и столбцов. Один такой «прямоугольник» называется страницей, а совокупность страниц называется банком.

- Элементы хранения (DRAM/SRAM)



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

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

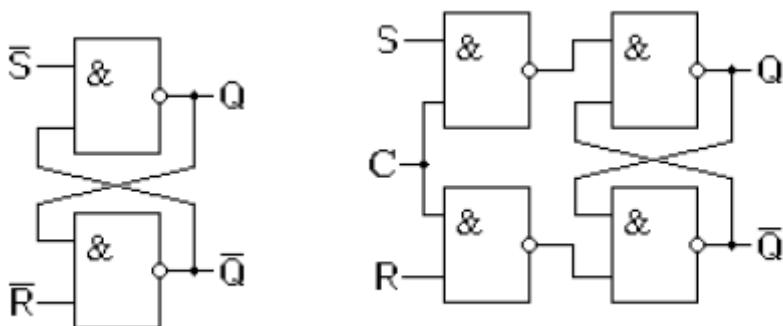
Вентильные схемы – это электронные ключевые схемы, предназначенные для управления потоком информации из регистров в шину и обратно. Такая схема содержит два входа и один выход. На один вход подается информационный сигнал (данные с регистра), а на другой (являющийся вентилем) – управляющий. Если управляющий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

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

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

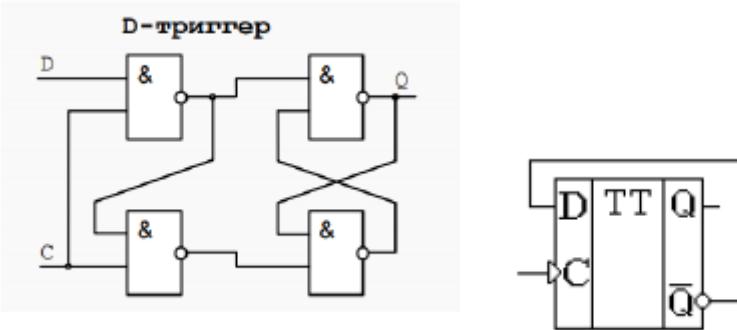
Триггер — класс электронных устройств, обладающих способностью длительно находиться в одном из двух устойчивых состояний и чередовать их под воздействием внешних сигналов. Каждое состояние триггера легко распознается по значению выходного напряжения. Отличительной особенностью триггера как функционального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

RS-триггер получил название по названию своих входов. Вход S (Set — установить англ.) позволяет устанавливать выход Q в единичное состояние. (Устанавливать означает записывать логическую единицу). Вход R (Reset — сбросить англ.) позволяет сбрасывать выход Q (Quit — выход англ.) в нулевое



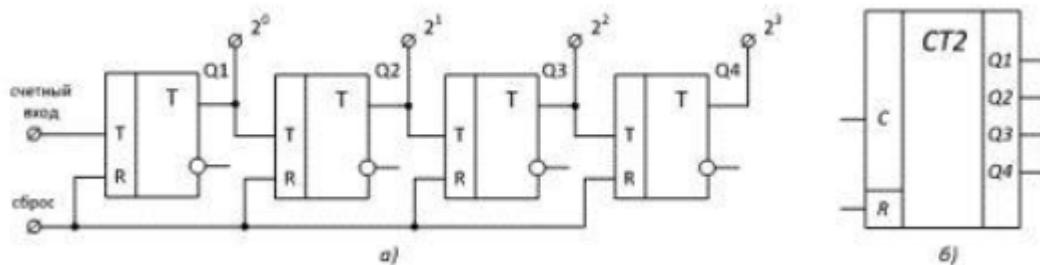
Синхронный RS-триггер. Схема, пропускающая входные сигналы только при наличии синхронизирующего сигнала. В этой таблице символ x означает, что значения логических уровней на данном входе не важны. Они не влияют на работу триггера.

D-триггер. При записи и хранении данных один бит может принимать значение, как нуля, так и единицы. Для его передачи достаточно одного провода. Сигналы установки и сброса триггера не могут появляться одновременно, поэтому можно объединить эти входы при помощи инвертора.



T-триггер — это счетный триггер. У данного триггера имеется только один вход. Принцип работы Т-триггера заключается в следующем. После поступления на вход Т импульса, состояние триггера меняется на прямо противоположное.

Счётчик числа импульсов — устройство, на выходах которого получается двоичный (двоично-десятичный) код, определяемый числом поступивших импульсов. Счетчики импульсов являются разновидностью регистров (счетные регистры) и строятся соответственно на триггерах и логических элементах. Основной параметр счётчика — модуль счёта — максимальное число единичных сигналов, которое может быть сосчитано счётчиком. На рисунке представлена схема четырехразрядного счетчика на Т-триггерах, соединенных последовательно. Счетные импульсы подаются на счетный вход первого триггера. Счетные входы последующих триггеров связаны с выходами предыдущих триггеров.



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

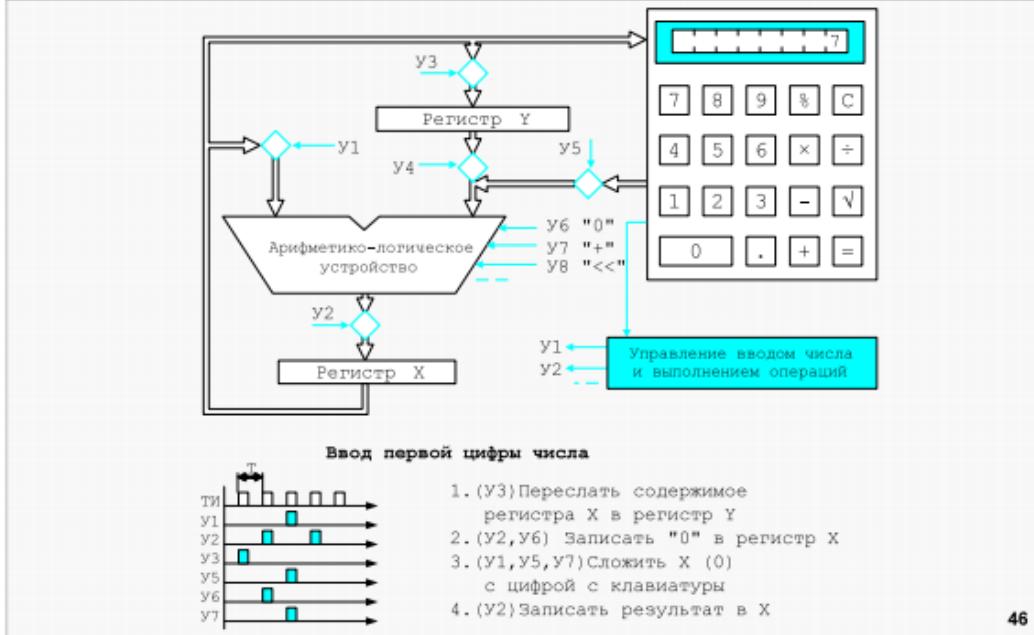
5. Структура и принцип функционирования ЭВМ. Порядок функционирования простого процессора на примере калькулятора.

Типичная ЭВМ состоит из процессора, памяти и устройств ввода-вывода.

«Сердцем» ЭВМ является **процессор**, в состав которого входят устройство управления выборкой команд из памяти и их выполнением, арифметико-логическое устройство, производящее операции над данными, регистры, осуществляющие временное хранение данных и состояний процессора, схемы для управления и связи с подсистемами памяти и ввода-вывода. Устройство ввода обеспечивает считывание информации с определенных носителей информации и ее представление в форме электрических сигналов, воспринимаемых другими устройствами ЭВМ.

Устройства вывода представляют результаты обработки информации в форме, удобной для визуального восприятия. Память ЭВМ включает устройство, обеспечивающее хранение команд и данных. Это устройство состоит из блоков одинакового размера – ячеек памяти, предназначенных для хранения одного слова информации.

Ячейка памяти состоит из элементов памяти, состояние каждого из которых соответствует одной двоичной цифре. Совокупность нулей и единиц, хранящихся в элементах одной ячейки, представляет собой содержимое этой ячейки памяти. В микро ЭВМ используются безадресные, одноадресные и реже двухадресные команды. В одноадресных командах один из операндов выбирается из специального регистра – аккумулятора. В него же заносится и результат операции. Безадресные команды или задают какое-либо действие с устройствами ЭВМ, или используются для работы с операндами, имеющими фиксированное расположение (чаще всего с аккумулятором). В процессе работы ЭВМ последовательно выполняет набор достаточно простых операций: выборку команды, определение ее типа, исполнение команды и определение адреса следующей команды.



Рассмотрим принципы функционирования простейшей ЭВМ — калькулятора. Он состоит из двух регистров — X и Y, хранящих результаты ввода пользователя и промежуточных вычислений, АЛУ, которая может выполнять простейшие арифметические и логические операции, шин и управляемых вентилей, осуществляющих передачу данных между функциональными блоками калькулятора, устройства управления (УУ), клавиатуры и дисплея.

Дисплей постоянно отображает содержимое регистра X (последите 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 управляемых импульсов для первой цифры числа.

В первую очередь необходимо сохранить предыдущее значение регистра X в регистре Y. Для этого должен быть открыт вентиль, управляющий записью в регистр Y. Он открывается управляемым сигналом У3.

После этого необходимо обнулить регистр X, подготовив его для новой цифры числа, которая была введена с клавиатуры. Для этого должны быть закрыты все вентили, кроме У2 — который осуществляет передачу данных из АЛУ в регистр Y 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.

6. Операционная система Unix — ядро ОС и файловая система

UNIX — семейство переносимых, многозадачных и многопользовательских операционных систем. Идеи, заложенные в основу UNIX, оказали огромное влияние на развитие компьютерных операционных систем. В настоящее время UNIX-системы признаны одними из самых исторически важных ОС. Основное отличие UNIX-подобных систем от других операционных систем заключается в том, что это изначально многопользовательские многозадачные системы. То есть в один и тот же момент времени сразу множество людей может выполнять множество вычислительных задач (процессов). Даже популярную во всём мире систему Microsoft Windows нельзя назвать полноценной многопользовательской системой, так как кроме как на некоторых серверных версиях, в один и тот же момент за одним компьютером с Windows может работать только один человек. В Unix может работать сразу много людей, при этом каждый из них может выполнять множество различных вычислительных процессов, которые будут использовать ресурсы именно этого компьютера. Вторая колossalная заслуга Unix в её мультиплатформенности. Ядро системы написано таким образом, что его легко можно приспособить практически под любой микропроцессор. UNIX имеет и другие характерные особенности:

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

Файловая система UNIX

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 уникальный полный путь. Каталог, являющийся корнем файловой системы (корневой каталог) имеет путь /. Коротким или относительным путем называется путь к файлу от текущего рабочего каталога. В каждом каталоге содержатся два специальных файла-ссылки, файл "." - ссылка на текущий каталог, и ссылка ".." на родительский каталог. inode - Index-node - описатель файла, его уникальный номер. Он содержит всю информацию о файле, за исключением имени файла, и собственно данных файла. В inode хранится: тип файла, права, время модификации/создания файла и другая служебная информация под общим названием «метаданные».

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

Командный интерпретатор – программа, предоставляющая пользователю интерфейс для общения с командной строкой; эта программа «переводит» введенные пользователем команды на понятный операционной системе язык. Интерпретатор более известен как оболочка (англ. shell). Наиболее распространенными оболочками являются sh, ksh, bash (стандарт в Unix), с 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 также стандартный поток ошибок, на который выводится вся служебная информация, которая не должна попадать в поток вывода (сообщения об ошибке или ходе работы программы). Стандартные потоки привязаны к файловым дескрипторам с номерами: 0 для ввода (stdin) 1 для вывода (stdout) 2 для ошибок (stderr).

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

Для осуществления перенаправления используются следующие операции:

Команда > файл (или >>)

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

Команда < файл

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

Команда1 | команда2

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

Команда 2> файл

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

Команда 2>&1 файл (или &> или >&)

Такой синтаксис используется для объединения потоков вывода и потока ошибок для обработки их вместе.

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

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

Основные команды

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.

In 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 категория.

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

r (read)

Право на чтение файла/просмотр каталога.

w (write)

Право на запись в файл/добавление или удаление каталога.

x (execute)

Право на исполнение файла/поиск и переход в каталог.

Права представляют собой последовательность из 9 бит – по 3 бита на категорию: владелец, группа, прочие; в следующем порядке – чтение, запись, исполнение. В случае отсутствия какого-либо из прав у категории, ставится символ «-».

Вторым способом записи прав является запись этой последовательности в 8-ричной системе счисления, где праву на чтение (r) соответствует цифра 4, праву на запись (w) – цифра 2, а праву на исполнение (x) – цифра 1. Цифра 0 означает отсутствие прав. Для получения конечной цифры, нужные права суммируются. Таким образом, запись занимает всего 3 бита: по 1 биту на категорию.

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

chmod [ugoa]{+-}[rwx] файл

Добавляет, удаляет или устанавливает выбранную комбинацию прав для выбранной комбинации категорий.

chmod число файл

Устанавливает права на основе восьмеричной записи.

chmod категория1=категория2 файл

Копирует права одной категории и присваивает их другой.

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

Базовая ЭВМ – это простая гипотетическая машина, обладающая типичными чертами многих конкретных ЭВМ. Память состоит из 2048 ячеек (16---битовых). Восемь ячеек памяти с адресами 008---00F называются индексными и их лучше использовать в циклических программах. Процессор состоит из ряда регистров (AC, BR, PS, IR, DR, CR, IP, SP, AR), ALU и устройства управления. Ячейки памяти с 000-00F выделяются под вектора прерывания.

AC- Аккумулятор.

BR-Почти всегда содержит адрес выполняемой команды, иногда промежуточные данные для некоторых команд.

PS-регистр состояния

0	C	Перенос
1	V	Переполнение
2	Z	Нуль
3	N	Знак
4	0	0 – используется для организации безусловных переходов в МПУ
5	EI	Разрешение прерываний
6	IRQ	Требование прерывания (логическое “И” шины запроса на прерывание и бита 5 РС – “разрешение прерываний”)
7	W	Состояние тумблеров РАБОТА/ОСТАНОВ (1 – РАБОТА)
8	P	Программа

IR-клавишный регистр

DR-Если адресная – значение ячейки к которой обращаемся, если безадресная – код команды(может использоваться для хранения промежуточных значений команды).

CR-всегда полный код команды

IP-всегда номер следующей команды.

SP- Указатель на ячейку стека.

AR-Если команда адресная – адрес ячейки к которой обращаемся, если безадресная – сам адрес команды.

ALU(арифметико - логическое устройство) - сложение, логического умножения, инвертирования, инкремента

Машина может одновременно выполнять арифметические и логические операции только с одним или двумя операндами. Один из операндов находится в AC, а второй – в DR. Результат помещается в AC. Регистр переноса(C) – это 1---разрядный регистр, выступающий в качестве продолжения аккумулятора и заполняющийся при переполнении AC. Этот регистр используется при выполнении сдвигов. АЛУ может выполнять такие арифметические операции, как сложение и вычитание с учетом переноса, полученного в результате выполнения предыдущей операции.

ЭВМ способна понимать и выполнять точно определенный набор команд. При составлении программы пользователь ограничен этими командами. В зависимости от того, к каким блокам базовой ЭВМ обращается команда или на какие блоки она ссылается, команды можно разделить на три группы: обращения к памяти (адресные команды), обращения к регистрам (регистровые или безадресные команды), команды ввода---вывода. Команды обращения к памяти предписывают машине производить действия с содержимым ячейки памяти, адрес которой указан в адресной части команды. Безадресные команды выполняют различные действия без ссылок на ячейку памяти. Команды ввода---вывода осуществляют обмен данными между процессором и внешними устройствами ЭВМ.

Безадресная команда

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0	0	0	0	Расширение КОП													

Команда ввода-вывода

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	Приказ				Устройство							

Команда ветвления

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Расш. КОП				Смещение							

Адресные команды бывают следующие:

... с прямой абсолютной адресацией

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
КОП				0	Адрес										

... с относительной адресацией

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
КОП				1	Режим				Смещение						

... с непосредственной загрузкой операнда

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
КОП				1	1	1	1	Число							

Эти действия (микрооперации) протекают в определенной временной последовательности и скоординированы между собой. Для обеспечения такой последовательности в ЭВМ используется ГТИ. Для реализации одной команды требуется выполнить определенное количество микрокоманд, каждой из которых инициируется одним тактовым импульсом. Общее число тактовых импульсов, требуемых для выполнения команды, определяет время ее выполнения, называемое циклом команды. Цикл команды обычно включает один или несколько машинных циклов. Устройство управления базовой ЭВМ может находиться в 5 возможных состояниях: выборки команды, выборки адреса, выборки операнда, исполнения, прерывания.

Цикл команды

- 1. Цикл выборки команды (Instruction Fetch, IF)
- 2. Цикл выборки адреса (Address Fetch, AF)
- 3. Цикл выборки операнда (Operand Fetch, OF)
- 4. Цикл исполнения (Execution, EX)
- 5. Цикл прерывания (Interrupt, INT)

Циклы пультовых операций

- Ввод адреса (Set Instruction Pointer, SIP)
- Чтение (Read, RD)
- Запись (Write, WR)
- Пуск (Start, ST)

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

Целые двоичные числа без знака можно использовать для представления нуля и целых положительных чисел. В 16-разрядном слове они могут изменяться от 0 до 65535. Это числа с фиксированной запятой. Целые двоичные числа со знаком используются, когда необходимо различать положительные и отрицательные числа. Отрицательные числа представляются в дополнительном коде. Это упрощает конструкцию ЭВМ. Сложение целых двоичных чисел со знаком и без знака выполняется в базовой ЭВМ с помощью команды ADD. По команде INC к содержимому аккумулятора прибавляется единица, а по команде DEC – единица вычитается. Если при этом возникает перенос из старшего разряда A, 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. Побитовая обработка данных обеспечивается командами логического умножения, циклических сдвигов, а также командами инвертирования и очистки регистра переноса. Команда AND выполняет над каждым разрядом аккумулятора и содержимым ячейки булеву операцию «И». Результат выполнения команды для каждой пары битов операндов равен 1 только тогда, когда оба бита равны 1, а в остальных случаях бит результата равен 0, т.е. команда позволяет выделять или очищать определенные биты слова. Команды ROL и ROR замыкают аккумулятор и регистр переноса в кольцо и сдвигают все биты кольца влево или вправо. Сдвигами числа можно реализовать операции умножения или деления на 2 (один сдвиг), 4 (два сдвига), 8 (три сдвига) и т.д. Выборка команды: 1) IP ---> BR, AR, 2) BR+1 -> IP 3) MEM(AR) ---> DR, 4) DR ---> CR, 4) Определение типа команды, вида адресации, 5*) Выполнение безадресных команд и команд ввода---вывода.

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

Символьные данные представляются в БЭВМ в виде 8-разрядного знакового числа, которое формируется с использованием той или иной кодировки. Строковые данные формируются из массива символьных данных: т.к. аккумулятор и ячейки в памяти БЭВМ 2-х байтовые, то максимально мы можем уместить всего 2 символа в каждую ячейку памяти.

Есть два типа массивов, реализуемых в БЭВМ:

- 1) Массив, хранящий в себе количество обрабатываемых ячеек - выбирается индексная ячейка, содержащая адрес первого элемента массива. Дальше либо записывается число обрабатываемых ячеек в доп. коде в любую выбранную ячейку, либо первый элемент массива переводится в доп. код, пересыпается в какую-нибудь ячейку, а потом мы работаем с ней через LOOP
- 2) Массив, содержащий в себе некоторый заданный стоп-символ - выбирается индексная ячейка, содержащая адрес первого элемента массива. Дальше поэлементно смотрим на каждый элемент массива, пока не встретим стоп-символ

Режимы адресации в бэвм. Цикл выборки адреса:

Код				Мнемоника	Название	Реализация
11	10	9	8			
0	M	M	M	ADD 0A / \$L	Прямая абсолютная	CR → DR, DR → AR; MEM(AR) → DR
1	0	0	0	ADD (L)	0x8 Косвенная относительная	CR(0..7) → BR, BR + IP → AR, MEM(AR) → DR, DR → AR; MEM(AR) → DR
1	0	1	0	ADD (L)+	0xA Косвенная автоинкрементная (постинкремент)	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)	0xB Косвенная автодекрементная (предекремент)	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 / (SP+N)	0xC Со смещением (sp)	CR(0, 7) → BR, BR + SP → DR, DR → AR; MEM(AR) → DR
1	1	1	0	ADD L / (IP+N)	0xE Прямая относительная	CR(0..7) → BR, BR + IP → DR, DR → AR; MEM(AR) → DR
1	1	1	1	ADD #N	0xF Прямая загрузка	CR(0, 7) → BR, BR → DR

Цикл выборки операнда: DR → AR, MEM(AR) → DR

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

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

Ветвления в программах организуются с помощью команд перехода. Они не изменяют содержимое аккумулятора и регистра переноса, а лишь изменяют содержимое IP, помещая в него адрес, определяемый адресной частью команды.



Команды ветвлений

ИТМО ВТ

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	BHS 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

Циклические программы используются в тех случаях, когда требуется несколько раз выполнить набор одинаковых действий над некоторым изменяющимся набором данных. Для организации циклических программ часто используют команды перехода и команду LOOP, которая служит для увеличения на 1 содержимого адресуемой ячейки памяти и перехода к одному из двух путей продолжения программы в зависимости от знака этого содержимого (если содержимое меньше 0, то выполняется команда, следующая за LOOP, в противном случае, эта команда пропускается, т.е. выполняется команда через одну за LOOP).

Также для циклов часто используют косвенную адресацию индексных ячеек. Если произвести косвенную адресацию какой-либо из индексных ячеек, то сначала ее содержимое будет использовано в качестве адреса операнда, а затем оно автоматически увеличится на 1.

DR после м.ц. OF содержит значение операнда

AR адрес операнда

- $\sim 0 + DR \rightarrow DR$; $\overline{0x0} = 0xFFFF = -1$; Вычитание единицы из DR
- $DR \rightarrow MEM(AR)$; Записываем значение операнда в память
- $\sim 0 + DR \rightarrow DR$; Вычитание еще единицы (ЗАЧЕМ?!!)
- **if** $DR(15) = 0$ **then** GOTO INT ; Проверка на положительный (DR-1) и если да, то завершение цикла
- IP + 1 → IP ; Перескок через команду, если DR-1 отрицательное
- GOTO INT ; Завершение цикла

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

Достаточно часто встречаются ситуации, когда отдельные части программы должны выполнить одни и те же действия по обработке данных. В подобных случаях повторяющиеся части программы выделяют в подпрограмму. В базовой ЭВМ для этой цели используется команда CALL. При оформлении подпрограммы перед ее первой командой следует разместить команду PUSH, в которую будет пересыпаться адрес возврата из подпрограммы. В команде обращения к подпрограмме указывается адрес именно этой ячейки, и команда CALL M выполняет следующие действия:
передает управление команде, расположенной по адресу M+1
Последней командой подпрограммы должна быть команда выхода (команда RET). По ней осуществляется выход из подпрограммы.



Цикл исполнения CALL

CALL: DR после м.ц. OF содержит адрес перехода

- 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

СТЕК – память типа LIFO. «Последним вошел - первым вышел» - LIFO - "Last In - First Out"
Обмен информацией между ЭВМ и стеком всегда выполняется через верхнюю ячейку – вершину стека. Стековая память очень удобна для упрощения решения многих задач, возникающих при работе с подпрограммами, обслуживания прерываний и т.д.

В БЭВМ нет аппаратной реализации стека, но его можно смоделировать. При этом в качестве стека используют просто часть адресной памяти. LIFO обеспечивается с помощью указателя стека (SP), который содержит адрес плавающей вершины стека.

Стек предусматривает 2 операции: загрузка нового слова (уменьшается содержимое SP, и слово записывается в ячейку, на которую он указывает) и извлечение слова из стека (читается содержимое ячейки, на которую указывает SP, а потом он увеличивается).

Push: $--SP, MOV$

Pop: $MOV, SP++$

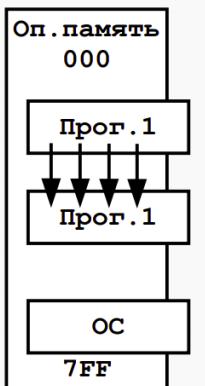
Варианты передачи данных в подпрограмму:

- Аккумулятор (Регистры Общего Назначения)
- Адресуемые ячейки памяти
- Стек – нет в БЭВМ, но можно реализовать (делать этого я, конечно, не буду)



PIC - Position Independent Code (перемещаемый код)

- Код, который работает относительно 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



Длина перемещаемой программы в БЭВМ?

39



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

- Набор стандартных библиотечных функций
- Разделяемые (динамически линкуемые) и архивные (статически линкуемые)
 - ```
find /lib /usr/lib -name "*.so" | wc -l
```

3510
  - Статические связывают вызовы функций с телом функции в процессе компиляции
  - Динамические — в момент загрузки
- Если вам нужна функция — см. в библиотеки

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



### Организация ввода-вывода в вычислительных системах

В вычислительных системах организация ввода-вывода реализуется за счёт шины ВУ, к которой подключаются контроллеры ВУ. Это сделано так потому, что ВУ гораздо медленнее, чем процессор, поэтому, если подключить ВУ напрямую к процессору, процессор будет тратить всё своё время на управление ВУ и ожидание его готовности, не выполняя более никакой полезной работы. Шина ВУ подразделяется на шину данных, шину управления и шину адреса. По шине адреса на контроллер приходят адреса регистров, по шине данных - данные для обмена, а по шине управления – управляющие сигналы, или приказы, которые указывают контроллеру, что именно нужно делать с полученной информацией. Обычно шину данных и шину управления не нужно использовать одновременно, поэтому их можно мультиплексировать – использовать одну и ту же шину для передачи управляющих сигналов в один момент времени и данных для обмена в другой.

Для обмена информацией используются драйверы.

### Драйвер

Драйвер – это специальная программа в памяти ЭВМ, которая реализует взаимодействие ВУ, процессора и ОЗУ. Она “знает” принцип работы ВУ, адреса регистров и поддерживаемые режимы работы. Эта программа содержит алгоритм обращения к ВУ и обработки данных поступающих с ВУ в процессор или с процессора в ВУ. Все драйверы управляются определённым единым программным интерфейсом, который задан операционной системой. Драйверы реализуют три стадии обмена: инициацию обмена, передачу данных и завершение обмена.

## **Инициация обмена**

Начало обмена информацией. На этом этапе драйвер должен вывести на шину управления приказ для ВУ. Есть три вида инициации обмена:

- Синхронный – драйвер отдаёт приказ на шину данных в определённый момент времени.
- Асинхронный – драйвер отдаёт приказ на шину данных только если устройство явно просигнализировало, что оно готово к обмену. Готовность проверяется самим процессором.
- Управляемая прерываниями – устройство сигнализирует процессору, что оно готово обмениваться данными. Отличие от асинхронного обмена состоит в том, что процессор сразу же приступает к обработке прерывания, то есть к обмену с ВУ, в то время как при асинхронном обмене процессор, получив готовность не обязан действовать в этот же момент.

## **Передача данных**

На этом этапе данные поступают из регистров контроллера ВУ в регистры процессора через шину данных. Есть два вида передачи данных:

- Асинхронная – процессор должен подтверждать получения каждого сегмента данных.
- Синхронная – процессор получает все данные без каких-либо дополнительных действий.

## **Завершение обмена**

На этом этапе на процессор приходит сигнал, что передача данных завершена. Есть два вида завершения обмена:

- Синхронный – оповещение о завершении обмена происходит сразу же после завершения работы драйвера.
- Асинхронный – оповещение о завершении обмена происходит в любой момент после завершения работы драйвера.

Ввод-вывод бывает программно-управляемый и управляемый аппаратурой.

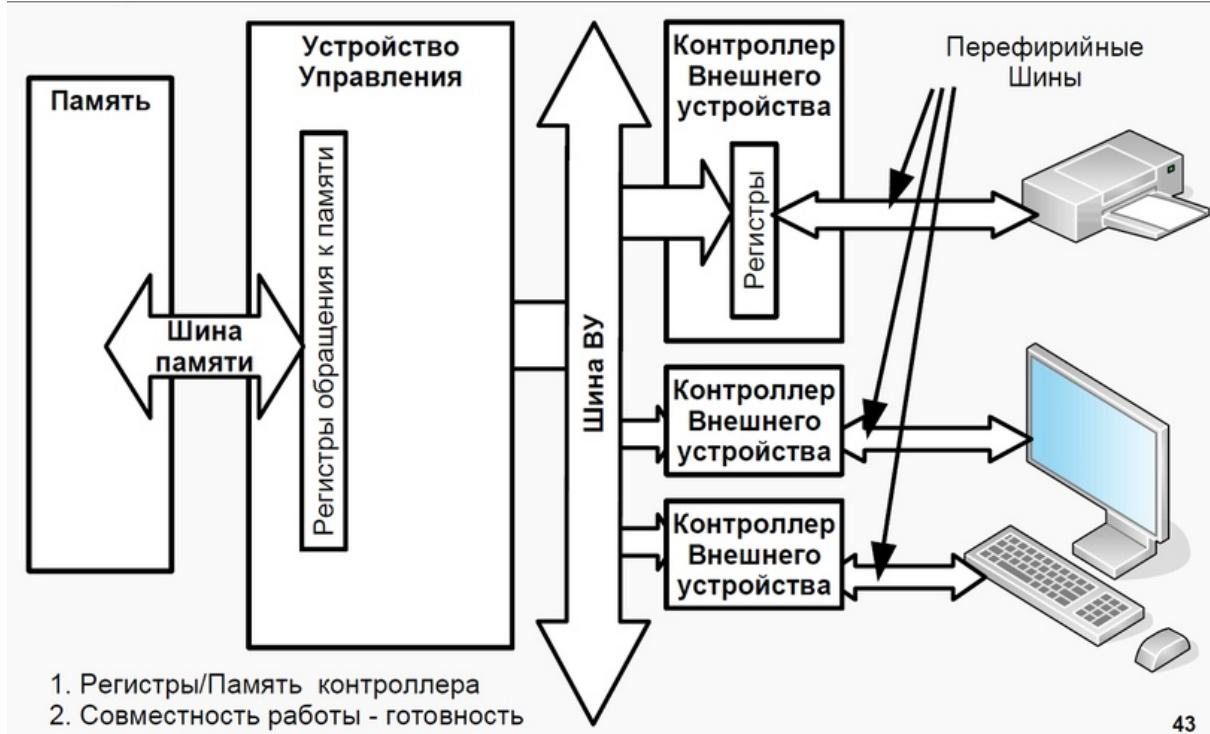
### **Программно-управляемый ввод-вывод**

Используется для обмена небольшими блоками информации с ВУ. Для обмена информацией с ВУ используются драйверы.

### **Управляемый аппаратурой ввод-вывод**

Используется для обмена большим объёмом данных между ВУ и ОЗУ. Так же используются драйверы, однако они нужны только для управления режимов работы контроллера, всё остальное делает сам контроллер. В процессе обмена процессор участия не принимает. Реализован при помощи специального контроллера прямого доступа в память (DMA), который самостоятельно инициирует обмен и транспортирует информацию с ВУ напрямую в предварительно подготовленные для этого ячейки памяти.

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



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 и управляемый прерываниями ввод-вывод, прямой доступ к памяти. Преимущества и недостатки

**Программно-управляемый ввод-вывод** - все действия, связанные с вводом-выводом управляются процессором. То есть ВУ работает с памятью через процессор. Главный минус подобной стратегии ввода-вывода: процессор простояивает в ожидании готовности ВУ. Процессор будет простоять до тех пор, пока ВУ не подаст сигнал о готовности передачи информации. Преимущества - легкость реализации.

**Управляемый прерываниями ввод-вывод** - работа ввода и вывода также происходит через процессор. Но теперь во время ожидания получения сигнала готовности ВУ передавать информацию, процессор может выполнять какую-либо полезную работу. Как только ВУ готова к обмену - она посылает сигнал готовности и просит прерывание (interruption) работы процессора. Такой подход с точки зрения производительности более правильный, но он требует больших усилий на обработку прерываний

**Прямой доступ к памяти (Direct Memory Access или DMA)** — ВУ работает с памятью напрямую, избегая процессор. Такой способ не нагружает процессор, следовательно ,

намного быстрее. Но для реализации данного подхода требуются более сложные контроллеры, что значительно удорожает конструкцию.

**Какие режимы передачи данных и управления вводом-выводом реализуемы в БЭВМ?**

**Почему не возможно реализовать другие?**

В БЭВМ реализованы

1.программно-управляемый ввод-вывод и управляемый прерываниями.

Программно-управляемый ввод-вывод достигается с помощью ожидания непосредственно в аккумуляторе бита статуса. Управляемый прерываниями достигается с помощью системы прерываний.

**Прямой доступ к памяти не реализован в БЭВМ**, так как нет соответствующих схем в контроллере ВУ!!!

## Регистры контроллера

**SR** — однобитовый статусный регистр, в котором регистрируется готовность ВУ к обмену данными. Содержит 1 когда готов, 0 — не готов. При извлечения его содержимого в АС будет находиться на месте 6 бита (то есть IN SR вернет либо 0000 0000, либо 0100 0000).

**MR** — 4-битовый регистр управления. Содержит в себе номер вектора прерывания и бит разрешения прерывания.

**DR** — 8-битовый регистр данных. Через него идет обмен данными. При извлечении его содержимого в АС занимает младший байт, не трогая старший.

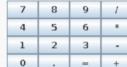
## Команды

### Команда ввода-вывода

|    |    |    |    |        |    |   |   |                    |   |   |   |   |   |   |   |
|----|----|----|----|--------|----|---|---|--------------------|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11     | 10 | 9 | 8 | 7                  | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0  | 0  | 0  | 1  | Приказ |    |   |   | Регистр устройства |   |   |   |   |   |   |   |

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

### Устройства:

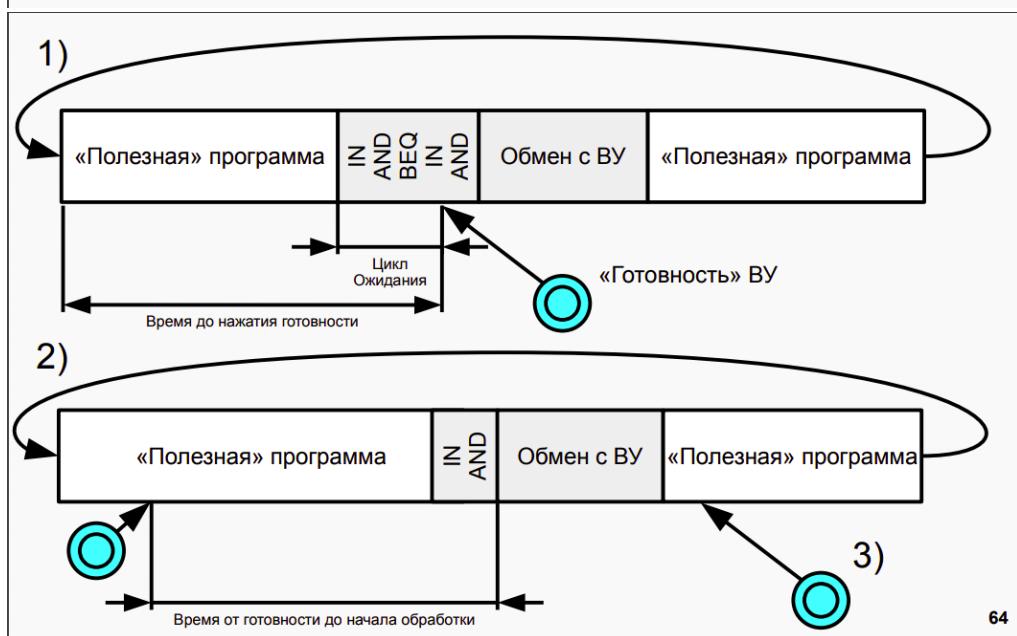
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>БЭВМ: ВУ-5 Текстовый принтер (регистры 0xC - 0xF)</b></p> <ul style="list-style-type: none"> <li>Печатает символы из РДВУ в заданной кодировке</li> <li>Регулируемая задержка (время печати) от 100 мс до 10 с</li> <li>Перевод строки по символу CR (0A16)</li> <li>NUL (0) — очистка</li> <li>Остальные - неопределенное поведение</li> </ul>                                                                                                                                                                                                                                       | <p><b>БЭВМ: ВУ-6 Бегущая строка (регистры 0x10 - 0x13)</b></p> <ul style="list-style-type: none"> <li>Размер матрицы: 32x8</li> <li>Сдвиг при записи нового значения в РДВУ</li> <li>Новое значение — справа</li> <li>Младший бит — нижний</li> </ul>                                                             | <p><b>БЭВМ: ВУ-7 8-ми разрядный 7-сегм. индикатор (0x14 - 0x17)</b></p> <ul style="list-style-type: none"> <li>Формат РДВУ: 7 654 3210<br/>0 Поз Симв.</li> <li>Симв==(A16) — установка в разряде знака «»</li> <li>Симв==(B16-F16) — сброс разряда</li> </ul>                                                        |
| <p><b>БЭВМ: ВУ-8 клавиатура (регистры 0x18-0x1C)</b></p> <ul style="list-style-type: none"> <li>Код нажатой клавиши в выбранной кодировке устанавливается в РДВУ</li> <li>Автоматически устанавливается готовность</li> </ul>                                                                                                                                                                                                                                                                            | <p><b>БЭВМ: ВУ-9 Цифровая клавиатура (0x1C-0x1F)</b></p> <ul style="list-style-type: none"> <li>При нажатии клавиши ее код помещается в РДВУ</li> <li>Клавиша 0-9 код 0-9</li> <li>Клавиша «-» код A</li> <li>«+» код B</li> <li>«/» код C</li> <li>«*» код D</li> <li>«.,» код E</li> <li>«=» код F</li> </ul>  | <p><b>БЭВМ: ВУ-0 Таймер (регистры 0x0, 0x1)</b></p> <ul style="list-style-type: none"> <li>Устанавливает готовность (и вызывает прерывание) раз в 100*DR миллисекунд       <ul style="list-style-type: none"> <li>Смещенная десятичная фиксированная точка</li> </ul> </li> <li>Если DR==0 готовность не устанавливается</li> <li>Можно использовать для организации синхронного обмена (Как?)</li> </ul> |
| <p><b>БЭВМ: У-во ввода-вывода ВУ-4 (регистры 0x8 - 0xB)</b></p> <ul style="list-style-type: none"> <li>По функционалу похоже на ВУ-3</li> <li>Адресуется 4-мя регистрами</li> <li>Отдельные регистры для входных (0x8 и 0x9) и выходных (0x9 и 0xA) данных</li> <li>Регистр состояния по адресу (0xA)       <ul style="list-style-type: none"> <li>Бит #0 все также отвечает за готовность ввода-вывода</li> </ul> </li> <li>Регистр управления по адресу (0xB)</li> <li>Все регистры доступны для чтения записи</li> <li>Позволяет реализовать сложные конфигурации подключения</li> </ul> | <p><b>Контроллер и устройство ввода-вывода ВУ-3</b></p>                                                                                                                                                                                                                                                          | <p><b>ВУ-2 «Устройство ввода»</b></p> <p>DR – 0x4</p> <p>SR, MR – 0x5</p> <p>Готовность устанавливается по нажатии клавиши готов на КВУ-2</p> <p>В DR нельзя писать. При попытке ничего не произойдет.</p> <p>SR недоступен для записи</p> <p>MR недоступен для чтения</p>                                           |

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

Асинхронный обмен - БЭВМ запрашивает готовность у КВУ, пока КВУ не ответит, что оно готово и не начнется передача данных.

| Ввод двух символов с устройства ввода ВУ-2 (DR#4, SR#5) |                                                                                                                                                                                                                                  |
|---------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ORG                                                     | 0x10                                                                                                                                                                                                                             |
| START:                                                  | CLA                                                                                                                                                                                                                              |
| S1:                                                     | IN 5 ; Ожидание ввода первого символа<br>AND #0x40 ; Бит 6 SR == 0 («Готов» нажата?)<br>BEQ S1 ; Нет - «Спин-луп»<br>IN 4 ; Ввод первого символа                                                                                 |
|                                                         | SWAB                                                                                                                                                                                                                             |
|                                                         | ST RES ; Сохранение его в ячейке RES                                                                                                                                                                                             |
| S2:                                                     | IN 5 ; Ожидание ввода второго символа<br>AND #0x40 ; Бит 6 SR == 0 («Готов» нажата?)<br>BEQ S2 ; Нет - «Спин-луп»<br>LD RES ; Ввод второго в младшие 8 разрядов А<br>IN 4 ; Ввод второго в младшие 8 разрядов А<br>ST RES<br>HLT |
| RES:                                                    | WORD ? ; Ячейка для записи слова "ДА"                                                                                                                                                                                            |

54



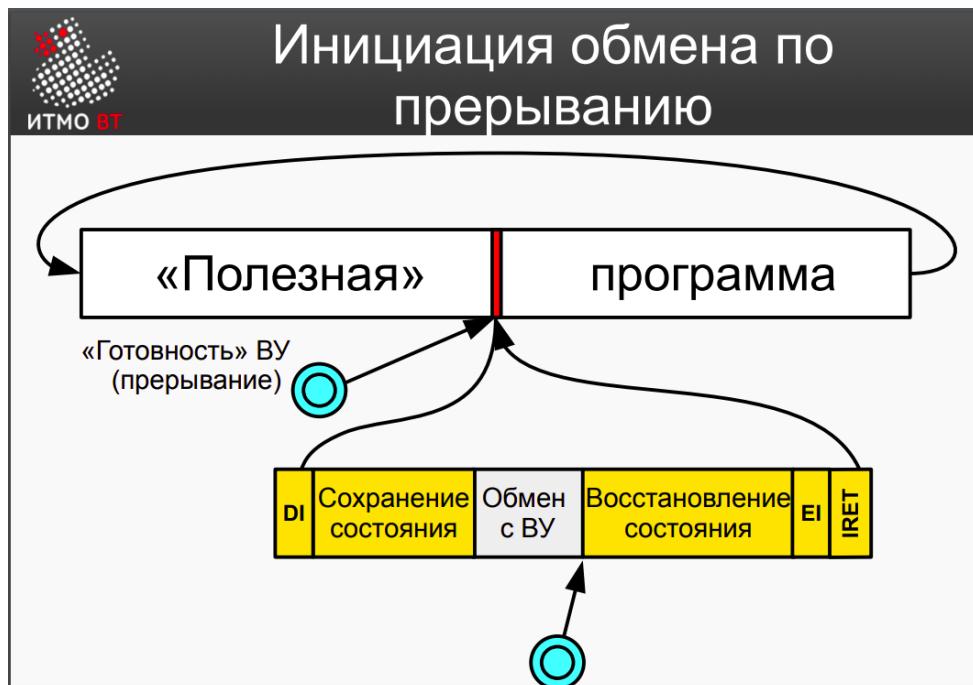
64

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

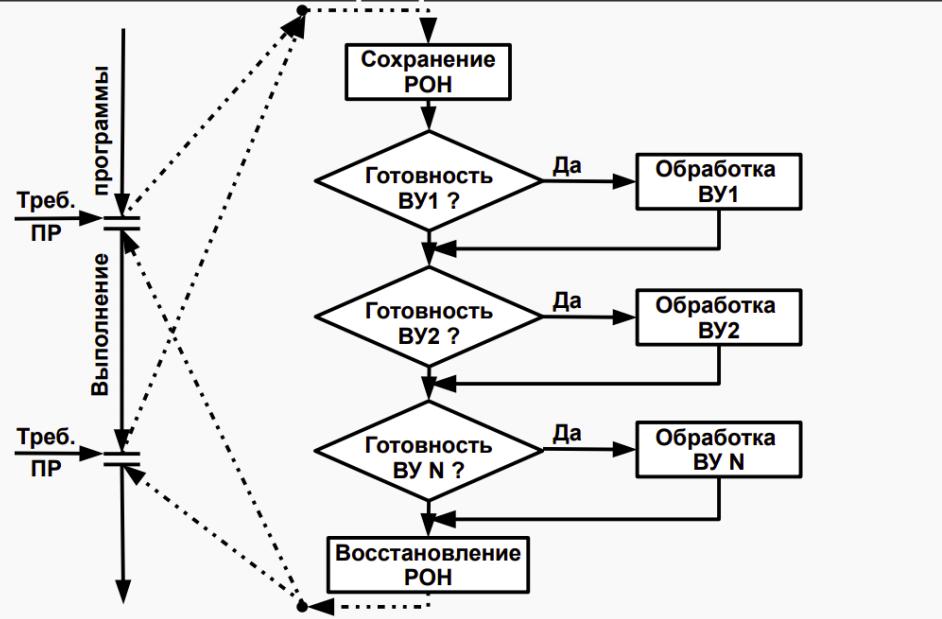
Прерывание — сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания. В контексте БЭВМ это сигнал о готовности обмена данными с некоторым ВУ.

Вектор прерывания — это совокупность адреса обработчика прерывания и регистра состояния, с которым этот обработчик стартует (данное высказывание справедливо в контексте БЭВМ, в других ЭВМ количество нужных ячеек для вектора прерывания варьируется).

В БЭВМ доступно 8 векторов прерываний, и расположены они в ячейках памяти 0x0 — 0xF включительно. На один вектор прерывания может приходится несколько прерываний.



## Логика обработки и приоритет: одно прерывание, много ВУ.

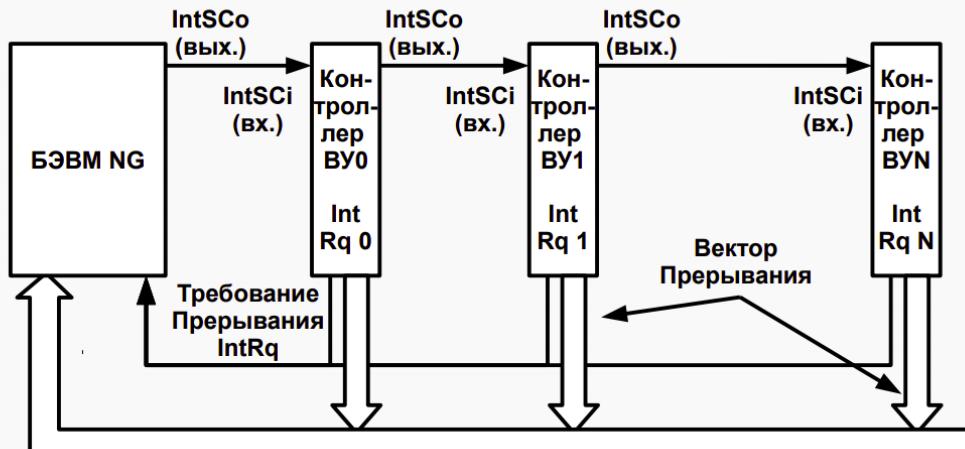


## Вектор прерывания

- 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

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

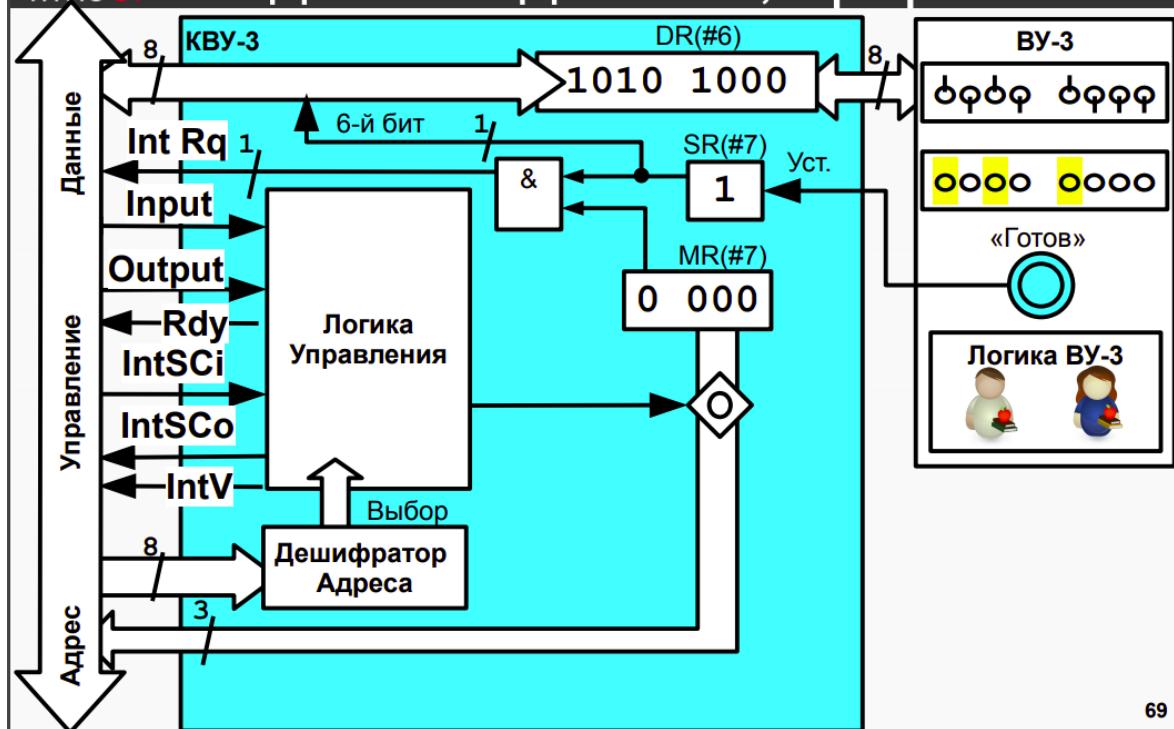
# Организация прерываний БЭВМ NG



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

68

## Контроллер и устройство ВВода-вывода ВУ-3, прерывания



69

| Сигнал (ы)                     | Направление<br>CPU . . . CNTRL                                                             | Описание                                                                                                                 |
|--------------------------------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>Addr0..7</b>                | <b>CR → Деш.адреса</b>                                                                     | Шина адреса контроллеров внешних устройств                                                                               |
| <b>Addr0..2</b>                | <b>CR ← MR контр.</b>                                                                      | Шина номера вектора прерывания (НВП)                                                                                     |
| <b>IntV</b>                    | <b>CPU ← л/у контр.</b>                                                                    | Сигнал подтверждения передачи в CR НВП                                                                                   |
| <b>Input</b>                   | <b>Деш.команд → л/у</b>                                                                    | Сигнал управления "Ввод"                                                                                                 |
| <b>Output</b>                  | <b>Деш.команд → л/у</b>                                                                    | Сигнал управления "Вывод"                                                                                                |
| <b>Rdy</b>                     | <b>CPU ← л/у</b>                                                                           | Сигнал подтверждения готовности                                                                                          |
| <b>IntRq</b>                   | <b>PS(IRQ) ← SR контр.</b>                                                                 | Запрос прерывания                                                                                                        |
| <b>IntSCI</b><br><b>IntSCo</b> | <b>CPU IRQSC → IntSCI0</b><br><b>IntSCo0 → IntSCI1</b><br><b>IntSCo1 → IntSCI2</b><br>.... | Цепочка сигналов предоставления прерывания.<br>Передается далее в случае отсутствия требования прерывания в контроллере. |
| <b>SYN</b>                     | <b>Такт.ген → л/у</b>                                                                      | Синхросигнал тактового генератора                                                                                        |
| <b>Data0..7</b>                | <b>AC ↔ рег. контр.</b>                                                                    | Двунаправленная шина данных связи АС с регистрами контроллера                                                            |

| Бит      | Мнем.      | Содержимое                                                                                              |
|----------|------------|---------------------------------------------------------------------------------------------------------|
| <b>0</b> | <b>C</b>   | Перенос                                                                                                 |
| <b>1</b> | <b>V</b>   | Переполнение                                                                                            |
| <b>2</b> | <b>Z</b>   | Нуль                                                                                                    |
| <b>3</b> | <b>N</b>   | Знак                                                                                                    |
| <b>4</b> | <b>0</b>   | 0 – используется для организации безусловных переходов в МПУ                                            |
| <b>5</b> | <b>EI</b>  | Разрешение прерываний                                                                                   |
| <b>6</b> | <b>IRQ</b> | Требование прерывания (логическое "И" шины запроса на прерывание и бита 5 РС – "разрешение прерываний") |
| <b>7</b> | <b>W</b>   | Состояние тумблеров РАБОТА/ОСТАНОВ (1 – РАБОТА)                                                         |
| <b>8</b> | <b>P</b>   | Программа                                                                                               |

## Команды для работы с прерываниями

|                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>IRET (0x0B00)</b>                                                                                                                                                                                                                                          | <b>DI (0x1000)</b>                                                                                                                                                                                                                                                      |
| <b>Возврат из прерывания</b>                                                                                                                                                                                                                                  | <ol style="list-style-type: none"> <li>Запрет прерываний</li> <li>Устанавливает 5 бит PS в 0</li> <li>Команда ввода-вывода</li> <li>-</li> <li><b>Будут выполняться:</b> IF, EXC, INT</li> <li><b>Не будут выполняться:</b> AF, OF</li> <li><u>Потактово</u></li> </ol> |
| <ol style="list-style-type: none"> <li>Возвращает состояние процессора до прерывания</li> <li>Безадресная команда</li> <li>-</li> <li><b>Будут выполняться:</b> IF, EXC, INT</li> <li><b>Не будут выполнятся:</b> AF, OF</li> <li><u>Потактово</u></li> </ol> |                                                                                                                                                                                                                                                                         |

## › EI (0x1100)

1. Разрешение прерываний
2. Устанавливает 5 бит PS в 1
3. Команда ввода-вывода
4. -
5. **Будут выполняться:** IF, EXC, INT
6. **Не будут выполнятся:** AF, OF
7. Потактово

1. Обрабатываются ли прерывания в пошаговом режиме (режиме останов) работы программы?

Почему?

В режиме останов — нет, так как цикл INT требует для отработки 1 в 7 бите PS.

В какой момент вызывается следующее прерывание, если пришло несколько запросов на прерывание (при PS 0x180)?

Как только отработал обработчик прерывания для первого прерывания, вызывается в самом его конце IRET. В EXC он вернет все на круги своя и в PS в 5 бите снова окажется 1 (EI) и потом в 6 бите 1 (IRQ) и цикле INT команды IRET начнется обработка следующего прерывания и так будет до тех пор, пока есть запросы на прерывание. Приоритет обработки я описывал выше.

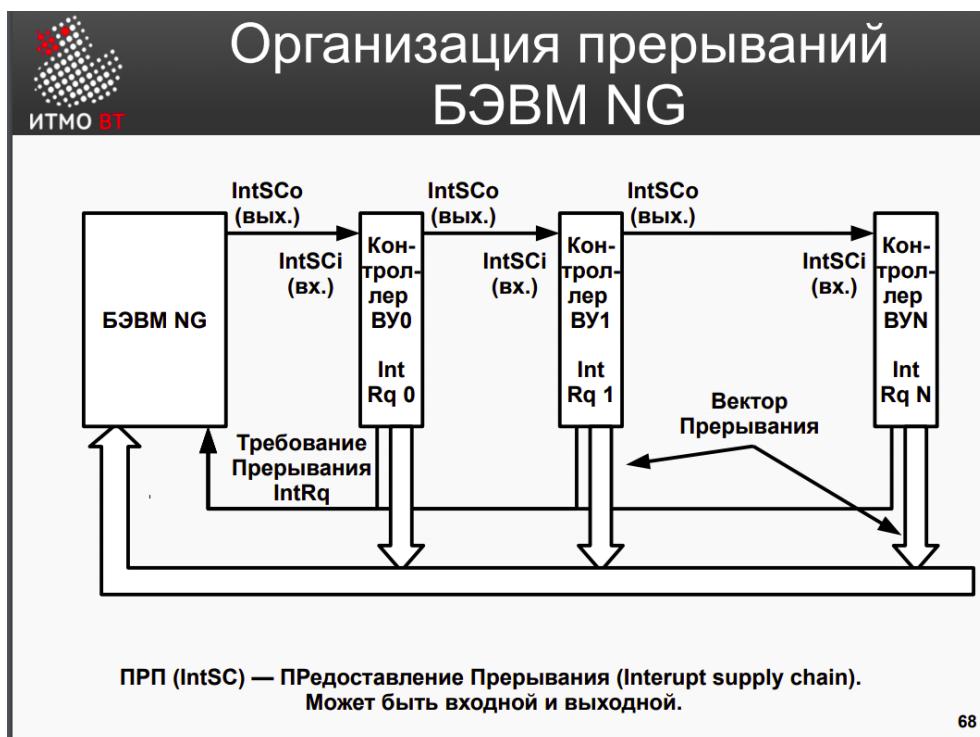
*Сигнал предоставления прерывания проходит через все КВУ, пока не дойдет до того, чье ВУ вызвало прерывания. Как именно КВУ понимает что именно его ВУ вызывает прерывание — секрет и спрятано в логике управления данного КВУ.*

*В БЭВМ приоритет получения КВУ сигнала предоставления реализован аппаратно, как Вы можете увидеть по картинке. Отсюда вытекает очень интересный факт, что приоритет обработки прерываний от разных ВУ реализован АППАРАТНО.*

*После всех команд (кроме HLT (0100)) идет цикл прерывания (INT) — именно в нем проверяется есть ли запрос на прерывания (при этом мы пока что не знаем какое ВУ требует прерывания)*

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

Используется при работе с низкоскоростными ВУ или когда момент передачи данных заранее неизвестен. Обмен данными между ЭВМ и ВУ инициируется сигналом с ВУ. Для реализации данного типа обмена используется аппаратная проверка наличия внешнего прерывания, т. е. сигнала готовности по линии "Запрос прерывания". По завершении цикла исполнения текущей команды происходит переход к циклу прерывания, который есть у всех команд кроме EI (Разр. прер.), DI (Запр. прер.) и HLT (Останов). Если в этот момент на линии «Запрос прерывания» нет сигнала о готовности ВУ или прерывания запрещены, то выполняется следующая команда. Иначе, прерывания запрещаются, в ячейку с адресом 000 заносится содержимое СК, и управление передается команде, расположенной в ячейке 001, с которой начинается программа обработки прерываний. Программа обработки прерываний запоминает в памяти содержимое А в ячейку SAVED\_A и содержимое С в ячейку SAVED\_C. Т.е. минимальная информация о прерванной программе хранится в ячейках 000, SAVED\_A и SAVED\_C. Производится поиск источника прерываний, и переход к последовательности действий по работе с конкретным ВУ. Затем выполняется передача данных и сброс флага готовности ВУ.



|                                                                |                                                                                                                                                                                                                                                                       |
|----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Готовность ВУ1: 2*A→РДВУ1, Готовность ВУ3: РДВУ3→ яч.3F</b> |                                                                                                                                                                                                                                                                       |
| <b>V0:</b>                                                     | ORG 0x0 ; Инициализация векторов прерывания                                                                                                                                                                                                                           |
| <b>V1:</b>                                                     | WORD \$DEFAULT,0x180 ; Вектор прерывания #0                                                                                                                                                                                                                           |
| <b>V2:</b>                                                     | WORD \$INT1,0x180 ; Вектор прерывания #1                                                                                                                                                                                                                              |
| <b>V3:</b>                                                     | WORD \$DEFAULT,0x180 ; Вектор прерывания #2                                                                                                                                                                                                                           |
| <b>V4:</b>                                                     | WORD \$INT2,0x180 ; Вектор прерывания #3                                                                                                                                                                                                                              |
| <b>V5:</b>                                                     | WORD \$DEFAULT,0x180 ; Вектор прерывания #4                                                                                                                                                                                                                           |
| <b>V6:</b>                                                     | WORD \$INT3,0x180 ; Вектор прерывания #5                                                                                                                                                                                                                              |
| <b>V7:</b>                                                     | WORD \$DEFAULT,0x180 ; Вектор прерывания #7                                                                                                                                                                                                                           |
| <b>DEFAULT:IRET</b>                                            | ; Просто возврат                                                                                                                                                                                                                                                      |
| ORG 0x020 ; Заргужка начальных векторов прерывания             |                                                                                                                                                                                                                                                                       |
| <b>START:</b>                                                  | DI<br>CLA<br>OUT 1 ; MR КВУ-0 на вектор 0<br>OUT 5 ; MR КВУ-2 на вектор 0<br>LD #9 ; разрешить прерывания и вектор №1<br>OUT 3 ; (1000 0001=1001) в MR КВУ-1<br>LD #0xB ; разрешить прерывания и вектор №3<br>OUT 7 ; (1000 0011=1011) в MR КВУ-3<br>...<br>BR \$PROG |

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

Готовность ВУ1: 2\*А→РДВУ1, Готовность ВУ3: РДВУ3→ яч. ЗF

```
ORG 0x040
INT3: ; Прерывание сохранило содержимое PS
 NOP ; отладочная точка останова (NOP/HLT)
 PUSH ; АС кладем в стек
 CLA
 IN 6 ; РДВУ-З
 ST $103 ; сохранение
 NOP ; отладочная точка останова (NOP/HLT)
 POP ; АС вынимаем из стека
 IRET ; Возврат из обработки прерывания
```



## Цикл прерывания

- if PS(W) = 0 then GOTO STOP; Проверка тумблера работа-останов, стоп если останов
- if PS(IRQ) = 0 then GOTO INFETCH; Если нет прерывания, то на выборку след. команды
- IRQSC ; Сформировать сигнал предоставление прерывания
- ~0 + SP → SP, AR }  
• IP → DR } ; IP → -(SP)  
• DR → MEM(AR)

72



## Цикл прерывания (2)

- ~0 + SP → SP, AR }
- PS → DR } ; PS → -(SP)
- DR → MEM(AR) ; а также...  
LTOL(CR) → BR ; младшие 8 разрядов CR  
(номер вектора прерывания) записать в BR
- SHL(BR) → BR, AR ; Вычисляем адрес ячейки с переходом на подпрограмму обработки прерывания, как номер вектора \* 2



## Цикл прерывания (3)

- **MEM(AR) → DR**; адрес обработчика прерывания записать в DR ...
- **DR → IP**; ... а затем в IP
- **LTOI(BR + 1) → AR**; ... выбрать адрес следующей ячейки вектора прерывания, ограничивая результат 8-ю разрядами
- **MEM(AR) → DR**; содержимое PS обработчика прерывания записать в DR ...
- **DR → PS**; ... а затем установить его в регистр

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

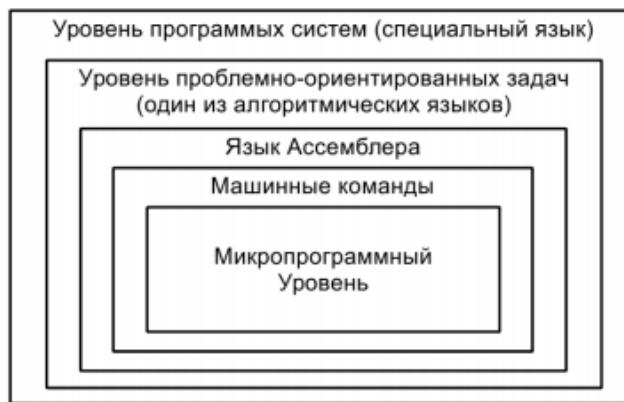
Возможность исполнения на ЭВМ программы, написанной на алгоритмическом языке, обеспечивается с помощью специальных системных программ: компиляторов и интерпретаторов. Компиляция, заключается в том, что процесс выполнения алгоритма осуществляется лишь после завершения процесса перевода исходной программы.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

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

Примером промежуточного языка служит язык символьического 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 символьических имен и меток. Компиляторы с таких языков называются ассемблерами. Они отводят определенные ячейки памяти для символьических переменных, организуют связи между различными частями программы, что резко облегчает программирование по сравнению с программированием на уровне команд.

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

Многоуровневая ЭВМ – это вычислительная машина, имеющая средства для работы с различными уровнями языков программирования. Нижний язык, или уровень, является наиболее простым, верхний – наиболее сложным. Такую машину можно рассматривать как в различных виртуальных машин, каждая из которых имеет свой машинный язык. Сложность аппаратурной реализации этих виртуальных машин возрастает по мере увеличения номера уровня.<sup>[6]</sup>



Рассмотрим примеры программ разных уровней.

### 1. Уровень программных систем.

В качестве примера можно привести сложение в экселе (прикладная программа).

X=Y+Z

X и Z лежат в ячейках, формулу помещаем туда куда надо - сами знаете как это работает.

Эксель преобразует то что мы сделали в язык, понятный более низким уровням машины.

## 2. Уровень проблемно-ориентированных задач.

Java - один из алгоритмических языков. Думаю что все, кто это читают, и без меня в состоянии написать программу сложения двух чисел на джаве.

## 3. Язык Ассемблера.

```
ORG ...
X: WORD...
Y: WORD...
Z: WORD...
LD Y
ADD Z
ST X
HLT
```

## 4. Машинные команды.

```
228 ... (переменная X)
229 ... (переменная Y)
230 ... (переменная Z)
231 A229 (LD Y)
232 4230 (ADD Z)
233 E228 (ST X)
234 0100 (HLT)
```

## 5. Микропрограммный уровень.

Тут слишком много придется писать, так что вот вам просто микрокоманда сложения:

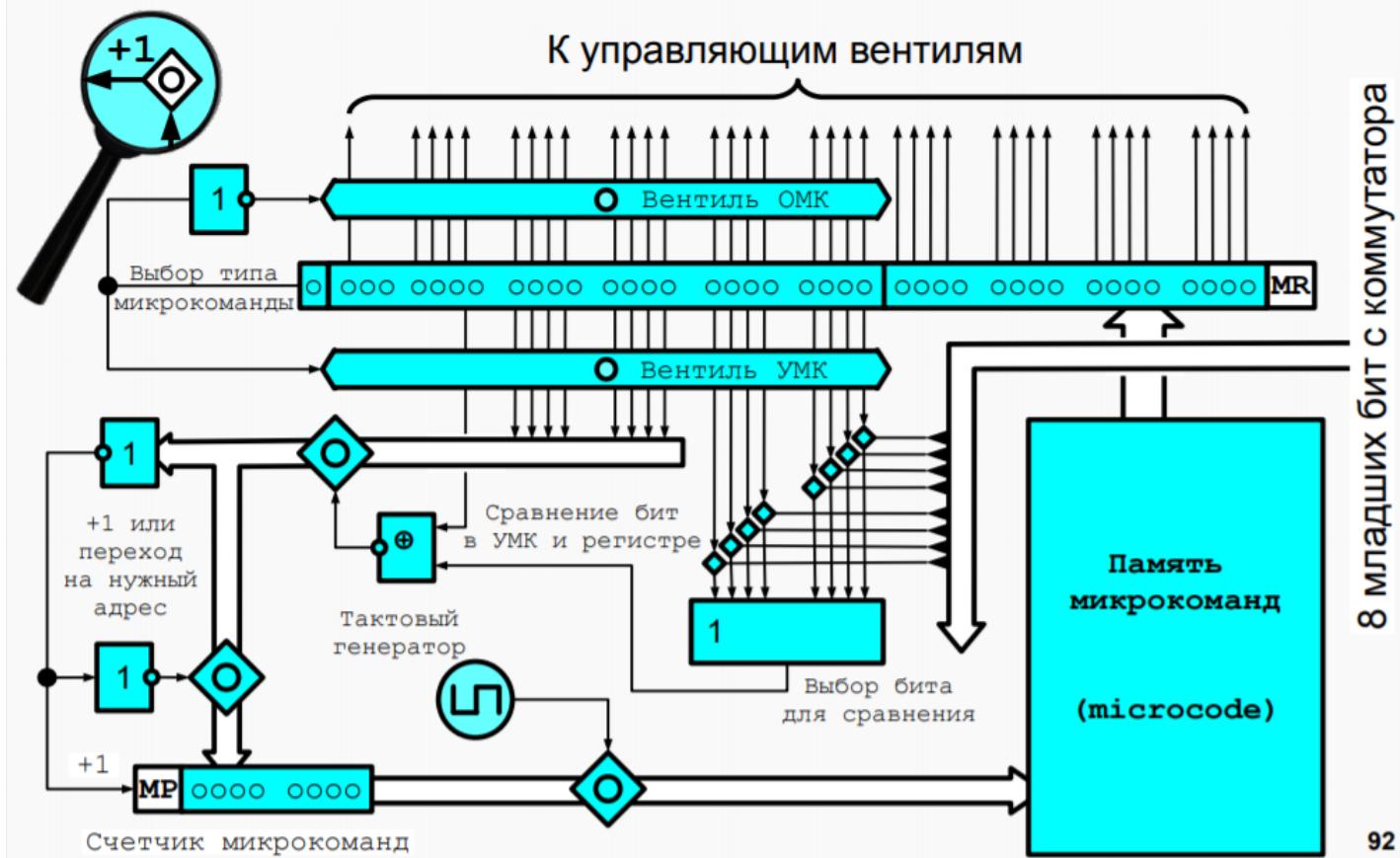
32 0010E09011 ADD AC + DR ? AC, N, Z, V, C

20. Микропрограммный уровень БЭВМ. Структура МПУ. Форматы микрокоманд

Если рассматривать БЭВМ как многоуровневую машину, то оказывается, что на её нижнем уровне выполняются элементарные действия (микрооперации) над словами информации. Управление порядком следования микроопераций осуществляется с помощью устройства управления БЭВМ, которое, в свою очередь, является очень простой ЭВМ. Для этой ЭВМ регистры и вентильные схемы базовой ЭВМ служат как бы устройствами ввода и вывода.

Программа работы такой ЭВМ – микропрограммного устройства управления (МПУ) – называется микропрограммой, а ее команды, содержащие информацию об элементарных действиях, подлежащих выполнению в течение одного рабочего такта ЭВМ, – микрокомандами. 29 Микропрограмма обычно хранится в постоянном запоминающем устройстве - памяти микрокоманд, но в эмуляторе БЭВМ реализована возможность изменения микропрограммы. В каждом такте работы ЭВМ из этой памяти в регистр микрокоманд (РМК) пересыпается очередная микрокоманда, т.е. микрокоманда, на которую указывает счетчик микрокоманд (СчМК), одновременно выполняющий функции регистра адреса микрокоманд.

**МПУ**  
Есть память микрокоманд, в счетчике +1, по его содержимому в памяти микрокоманд выбирается значение и поступает в регистр микрокоманд, в регистре выбирается тип команды и открывается определенный вентиль:  
 1) операционная все закодированные биты микрокоманды поступают на вентильные схемы  
 2) управляющая: часть вентильные, значение регистра в устройство управления , там оно сравнивается с установленными в микрокоманде) если условие выполнено то загружается счетчик микрокоманд новым значением, если нет то добавляется единица . Общий принцип устройства упр микрокоманд. //Если проверяемый бит и бит из поля сравнения идентичны, то схема сравнения формирует единичный сигнал, который открывает вентильную схему ВА и на СчМК передается адрес перехода (16-24-ый биты УМК). В противном случае на СчМК передается нулевое значение, которое инициирует увеличение значения СчМК на единицу (используется схема ИЛИ-НЕ для всех битов адреса перехода).



8 младших бит с коммутатора

92

MP нужна для того: если команда не операционная то будут нули, и эта схема блокирует запись нулей в счетчик микрокоманд.

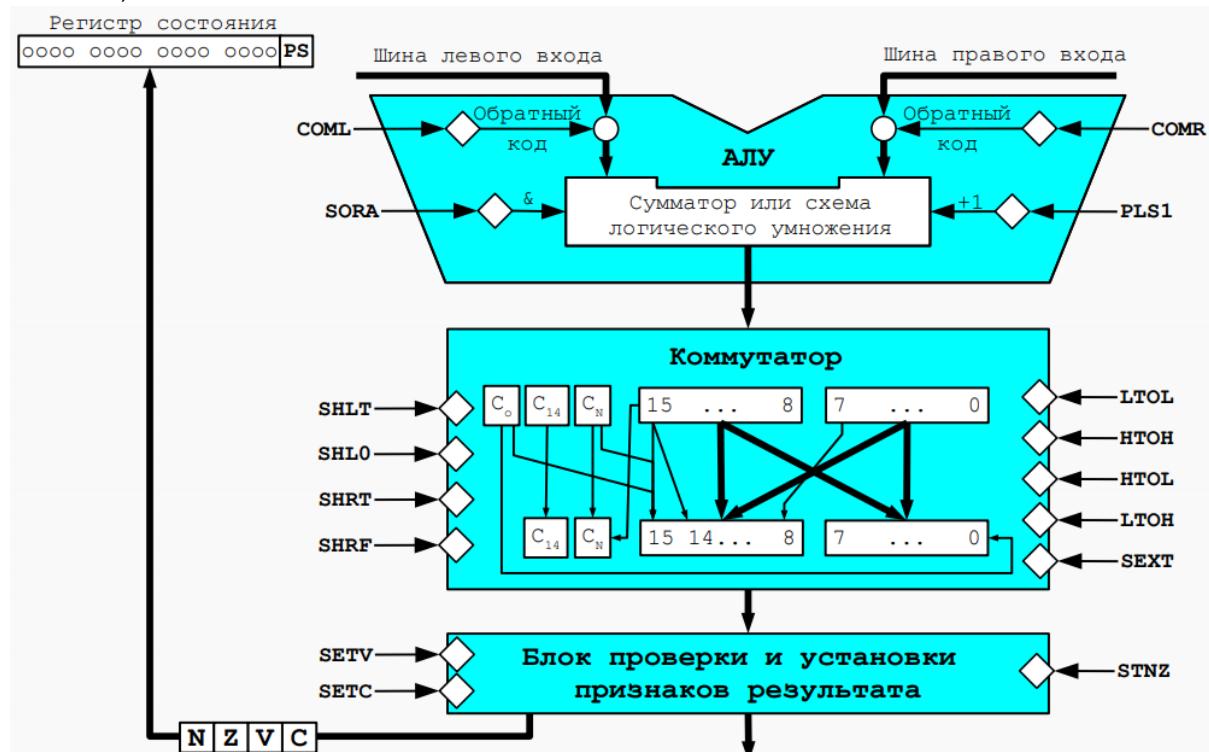
Microcode pointer используется для загрузки(выборки) микрокоманды из памяти и загрузки в регистр микрокоманд Вертикальная микрокоманда

Так как в ЭМВ больших размеров могут быть сотни вентильных схем горизонтальная команда может быть неприемлемо длинной, поэтому используют кодируемые поля. Кодируемые поля, шифруют вентили, которые не могут быть открыты одновременно. В управляющей микрокоманде сокращение её разрядов можно обеспечить за счет кодирования полей выбора проверяемого регистра и проверяемого бита в этом регистре, что позволяет сократить УМК до 16 бит. Однако при декодировании таких сжатых полей требуется использовать дешифратор.

Для декодирования операционных микрокоманд ОМК1 и ОМК2 потребуется ещё 8 дешифраторов. Но экономия затрат на память микрокоманд будет превышать стоимость этих дешифраторов.

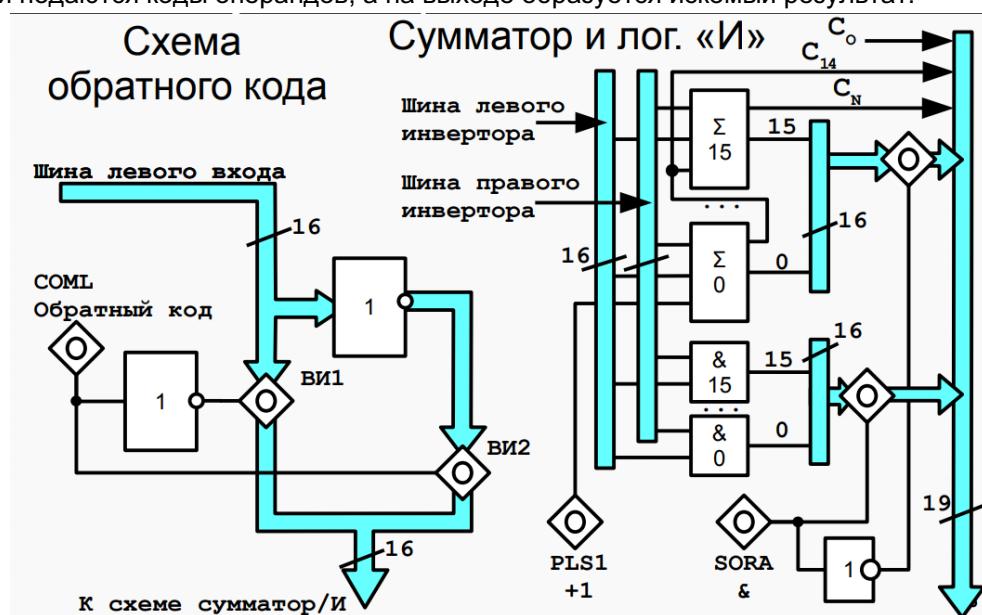
## 21. Структура и принципы работы арифметико-логического устройства и коммутатора. Регистр состояния БЭВМ

АЛУ использует А и РД в качестве операндов для получения результата, который помещается в аккумулятор. Все арифметические и логические команды БЭВМ и вспомогательные арифметические операции (например, увеличение на единицу содержимого СК) можно выполнить с помощью АЛУ.



Обратный код входных сигналов АЛУ получается по схеме, которая приведена слева снизу. В данной схеме изменение всех 0 на 1 и всех 1 на 0 осуществляется с помощью инверторов.

Сложение осуществляется типовой схемы сумматора, которая приведена справа. Логическое умножение осуществляется с помощью вентильной схемы, на входы которой подаются коды операндов, а на выходе образуется искомый результат.



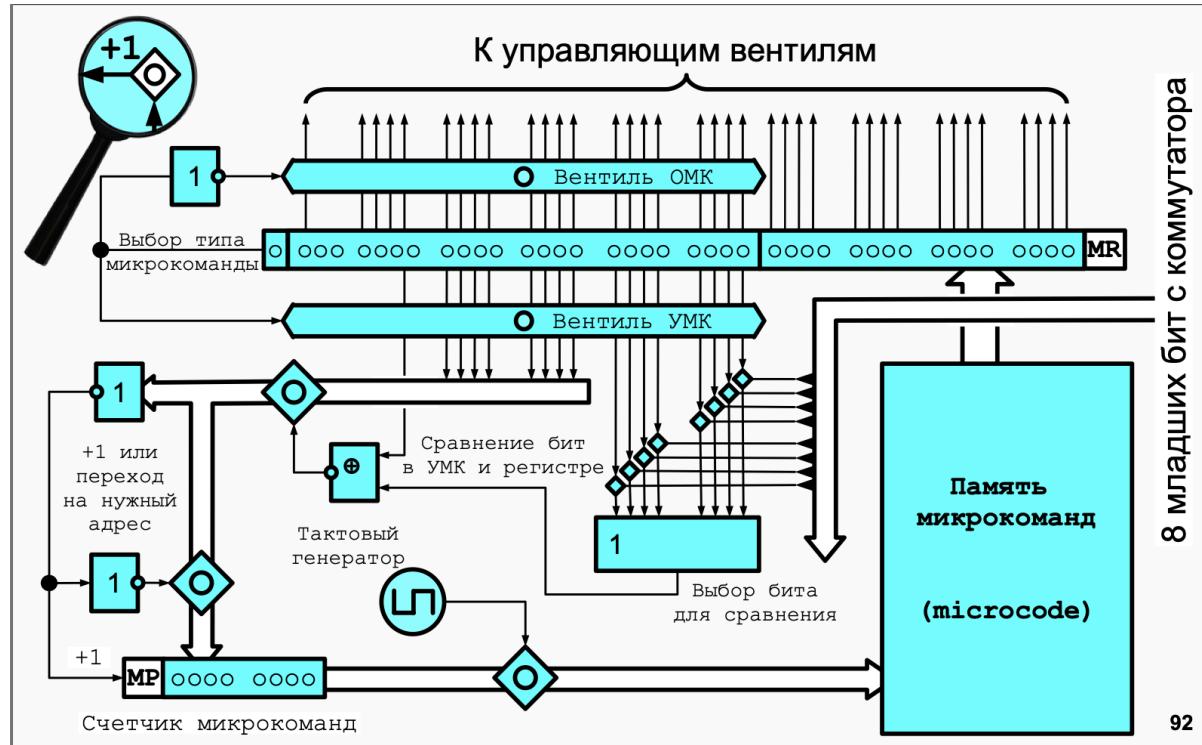
При организации ветвлений в микропрограмме используется содержимое РС, являющееся объединением однобитовых признаков результата и служебных битов состояния ЭВМ. Вентильные схемы – это электронные ключевые схемы, предназначенные для управления потоком информации из регистров в шину и обратно. На один вход подается информационный сигнал, на другой – управляющий.

Регистр состояния и команд(PS):

| Бит | Мнем. | Содержимое                                                                                              |
|-----|-------|---------------------------------------------------------------------------------------------------------|
| 0   | C     | Перенос                                                                                                 |
| 1   | V     | Переполнение                                                                                            |
| 2   | Z     | Нуль                                                                                                    |
| 3   | N     | Знак                                                                                                    |
| 4   | O     | 0 – используется для организации безусловных переходов в МПУ                                            |
| 5   | EI    | Разрешение прерываний                                                                                   |
| 6   | IRQ   | Требование прерывания (логическое "И" шины запроса на прерывание и бита 5 РС – "разрешение прерываний") |
| 7   | W     | Состояние тумблеров РАБОТА/ОСТАНОВ (1 – РАБОТА)                                                         |
| 8   | P     | Программа                                                                                               |

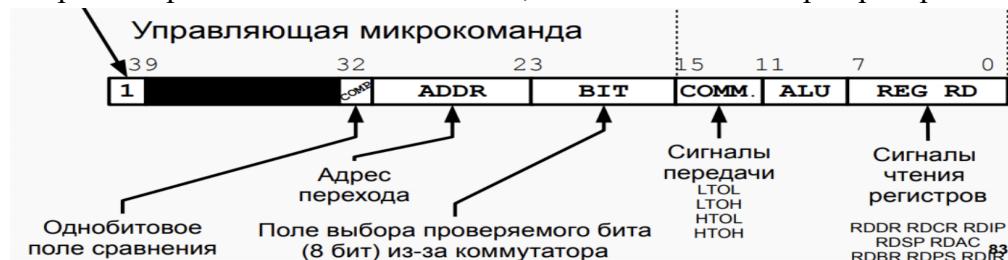
## 22. Микропрограммное управление вентильными схемами. Схема управления. Интерпретатор БЭВМ.

Для того, чтобы разобраться как происходит управление вентильными схемами, рассмотрим подробнее работу устройства управления.



Содержимое памяти микрокоманд в каждом такте (внимание на тактовый генератор слева) передается в регистр микрокоманд (MR, 40 разрядный). Младшие 16 разрядов (aka одинаковая часть независимо от типа микрокоманды) сразу передаются на управляющие вентили. Внимание на бит 39. Это код операции микрокоманды. Если он НУЛЯМБА, то микрокоманда операционная. Сразу под лупой (не за лупой) вы можете видеть инвертор. Предположим, что наш 39 бит - 0. Тогда он поступает на инвертор, на выходе инвертора получается единица. Открывается вентиль ОМК. Оставшиеся биты тоже сваливают на управляющие вентили. Все просто, все закончилось.

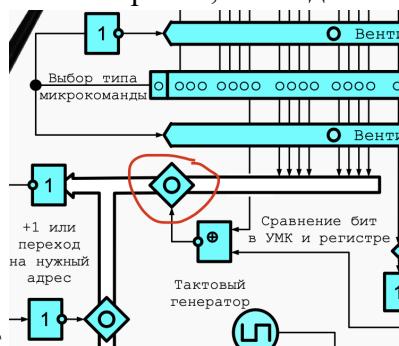
Предположим теперь, что код операции МК не нулямба, а единица. Откроется вентиль УМК. Среди тех бит, которые еще не ушли на вентильные схемы, рассмотрим младшие 8. Напомню, что это - поле выбора проверяемого бита.



Видите белую стрелочку которая тупа с правого куска картинки появляется? По ней приходят 8 младших бит с коммутатора. Видите мелкие ромбики? Это 8

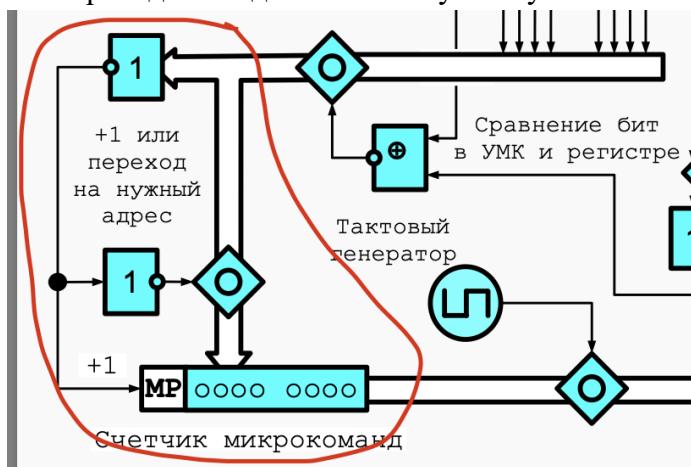
вентиляй. Если происходит совпадение одного из 8 битов МК и одного из 8 битов коммутатора, соответствующий вентиль открывается. Прямоугольник цвета циан (ну или тиффани) чуть ниже вентиляй - это одна большая схема ИЛИ. Результат идет на один из входов обратного ксора. На второй вход этого же самого обратного ксора идет бит номер 32. Посмотрев на картинку выше вспоминаем, что это “однобитовое поле сравнения”.

Брат, ты помнишь что такое обратный ксор? Он дает единичку, если два его входа совпадают. Таким образом, если два входа совпали, то открывается вентиль.



Какой? Этот.

Рассмотрим три случая. Случай первый: вентиль в самом деле открылся. Тогда адрес перехода попадает на вот эту схему:



Эта схемка нужна для того, чтобы понять, меняем ли мы MP, или просто прибавляем к нему АДЫН.

Адрес перехода не ноль? Не ноль. Смотрим в красную картошку. Верхний инвертор проверяет, не равен ли нулю наш адрес перехода. Если не равен, то открывается вентиль в красной картошке, адрес перехода записывается в MP. Ура!

Рассмотрим второй случай. Тот самый инвертированный ксор выдал на своем выходе ноль. Это значит, что адрес перехода не попал никуда дальше. Верхний инвертор нашей красной картошки получил на вход 0, выдал на выходе 1 и просто прибавил к MP единицу. Вентиль внутри картошки (если вы не догадались, так я называю красную “окружность” на картинке) не откроется, MP не заполнится нулями. Ура!

Рассмотрим третий случай. Наша команда была вообще говоря операционная изначально, а не управляющая. И снова, аналогично предыдущему случаю, наша

схема в красной картошке предотвращает запись нулей в МР, просто инкрементируя его вместо этого. Ура!

Мы разобрались как работает устройство управления. Идем дальше.

Чтобы понять, какой бит за что отвечает в твоей микрокоманде, достаточно лишь воспользоваться шпаргалкой вентильных схем.

| <b>Бит</b>              | <b>Мнемоника</b> | <b>Назначение</b>              |
|-------------------------|------------------|--------------------------------|
| <b>Чтение регистров</b> |                  |                                |
| <b>00</b>               | <b>RDDR</b>      | DR (РД) → правый вход АЛУ      |
| <b>01</b>               | <b>RDCR</b>      | CR (РК) → правый вход АЛУ      |
| <b>02</b>               | <b>RDIP</b>      | IP (СК) → правый вход АЛУ      |
| <b>03</b>               | <b>RDSP</b>      | SP (УС) → правый вход АЛУ      |
| <b>04</b>               | <b>RDAC</b>      | AC (A) → левый вход АЛУ        |
| <b>05</b>               | <b>RDBR</b>      | BR (БР) → левый вход АЛУ       |
| <b>06</b>               | <b>RDPS</b>      | PS (PC) → левый вход АЛУ       |
| <b>07</b>               | <b>RDIR</b>      | KR (KP) → левый вход АЛУ       |
| <b>АЛУ</b>              |                  |                                |
| <b>08</b>               | <b>COMR</b>      | Обратный код правого входа АЛУ |
| <b>09</b>               | <b>COML</b>      | Обратный код левого входа АЛУ  |
| <b>10</b>               | <b>PLS1</b>      | Операция A + B + 1 (PLuS 1)    |
| <b>11</b>               | <b>SORA</b>      | Операция И (Sum OR And)        |

94

| <b>Бит</b>                     | <b>Мнемоника</b> | <b>Назначение</b>                                                         |
|--------------------------------|------------------|---------------------------------------------------------------------------|
| <b>Управление коммутатором</b> |                  |                                                                           |
| <b>12</b>                      | <b>LTOL</b>      | Прямая передача младшего байта                                            |
| <b>13</b>                      | <b>LTON</b>      | Передача младшего байта в старший                                         |
| <b>14</b>                      | <b>HTOL</b>      | Передача старшего байта в младший                                         |
| <b>15</b>                      | <b>HTON</b>      | Прямая передача старшего байта                                            |
| <b>16</b>                      | <b>SEXT</b>      | Расширение знака младшего байта (sign extend)                             |
| <b>17</b>                      | <b>SHLT</b>      | Сдвиг влево (арифметический) (SHift Left)                                 |
| <b>18</b>                      | <b>SHLO</b>      | Передача старого значения С в младший бит (для циклического сдвига влево) |
| <b>19</b>                      | <b>SHRT</b>      | Сдвиг вправо (арифметический) (SHift Right)                               |
| <b>20</b>                      | <b>SHRF</b>      | Переключатель сдвига вправо (для циклического сдвига 15 разряд)           |
| <b>21</b>                      | <b>SETC</b>      | Установка С                                                               |
| <b>22</b>                      | <b>SETV</b>      | Установка V                                                               |
| <b>23</b>                      | <b>STNZ</b>      | Установка N и Z                                                           |

95

| Бит                     | Мнемоника   | Назначение              |
|-------------------------|-------------|-------------------------|
| <b>Чтение регистров</b> |             |                         |
| <b>24</b>               | <b>WRDR</b> | АЛУ → DR (РД)           |
| <b>25</b>               | <b>WRCR</b> | АЛУ → CR (РК)           |
| <b>26</b>               | <b>WRIP</b> | АЛУ → IP (СК)           |
| <b>27</b>               | <b>WRSP</b> | АЛУ → SP (УС)           |
| <b>28</b>               | <b>WRAC</b> | АЛУ → AC (A)            |
| <b>29</b>               | <b>WRBR</b> | АЛУ → BR (БР)           |
| <b>30</b>               | <b>WRPS</b> | АЛУ → PS (PC)           |
| <b>31</b>               | <b>WRAR</b> | АЛУ → AR (PA)           |
| <b>Работа с памятью</b> |             |                         |
| <b>32</b>               | <b>LOAD</b> | Ячейка памяти → DR (РД) |
| <b>33</b>               | <b>STOR</b> | DR (РД) → Ячейка памяти |

96

| Бит                             | Мнемоника   | Назначение                      |
|---------------------------------|-------------|---------------------------------|
| <b>Организация ввода-вывода</b> |             |                                 |
| <b>34</b>                       | <b>IO</b>   | Передача адреса и приказа на ВУ |
| <b>35</b>                       | <b>IRQS</b> | Предоставление прерывания       |
| <b>Резервные сигналы</b>        |             |                                 |
| <b>36</b>                       |             | Зарезервирован                  |
| <b>37</b>                       |             | Зарезервирован                  |
| <b>Останов БЭВМ</b>             |             |                                 |
| <b>38</b>                       | <b>HALT</b> | Останов                         |
| <b>Работа с памятью</b>         |             |                                 |
| <b>39</b>                       | <b>TYPE</b> | Бит выбора ОМК/УМК              |

А че такое интерпретатор?

Он состоит из этого:

- 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

99

А нафига ячейки зарезервированы хочешь спросить? А это для 7 лабораторной работы.

Чтобы изучить интерпретатор, посмотреть на него и потыкать, достаточно лишь

**java -Dmode=decoder -jar  
bcomp-ng.jar**

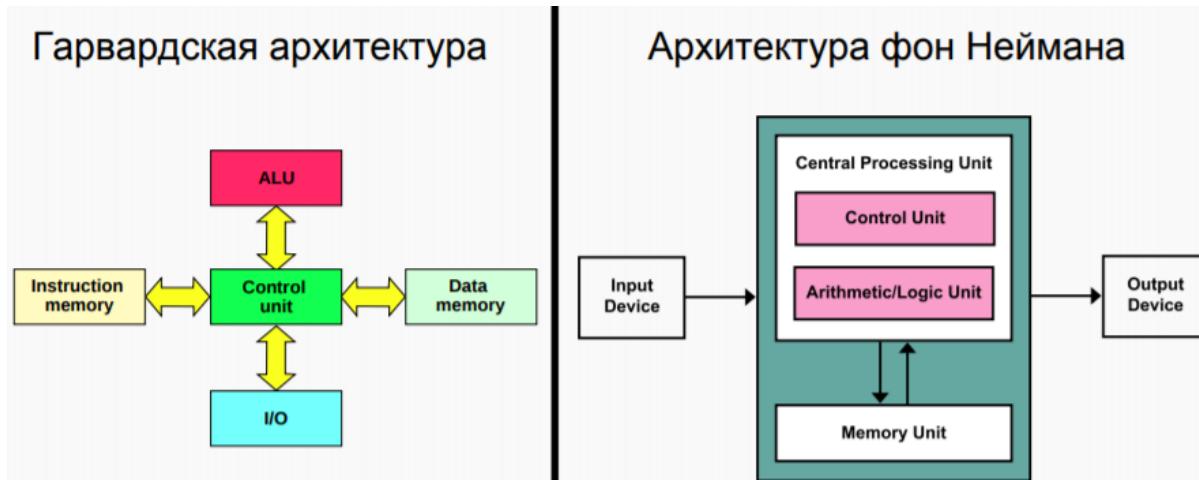
. ля какой шрифт большой

Рекомендую открыть и поискать глазками все блоки из предыдущей картинки, полезно для понимания.

В интерпретаторе есть 256 ячеек для хранения микрокоманд. Причем горизонтальных.

Правильное ударение: бИты!

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



Гарвардская архитектура характеризуется физическим разделением памяти команд и памяти данных. Каждая память соединяется с процессором отдельной шиной, что позволяет одновременно с чтением-записью данных при выполнении текущей команды производить выборку и декодирование следующей команды. Благодаря такому разделению потоков команд и данных и совмещению операций их выборки реализуется более высокая производительность, чем при использовании Принстонской архитектуры.

**Недостатки** Гарвардской архитектуры связаны с необходимостью проведения большего числа шин, а также с фиксированным объемом памяти, выделенной для команд и данных, назначение которой не может оперативно перераспределяться в соответствии с требованиями решаемой задачи. Поэтому приходится использовать память большего объема, коэффициент использования которой при решении разнообразных задач оказывается более низким, чем в системах с Принстонской архитектурой. Однако развитие микроэлектронной технологии позволило в значительной степени преодолеть указанные недостатки, поэтому Гарвардская архитектура широко применяется во внутренней структуре современных высокопроизводительных МП, где используется отдельная кэш-память для хранения команд и данных. В то же время во внешней структуре большинства микропроцессорных систем (МПС) реализуются принципы Принстонской архитектуры.

Принстонская архитектура, которая часто называется архитектурой Фон-Неймана, характеризуется использованием общей оперативной памяти для хранения программ, данных, а также для организации стека. Для обращения к этой памяти используется общая системная шина, по которой в процессор поступают и команда, и данные. Эта архитектура имеет ряд важных достоинств. Наличие общей памяти позволяет оперативно перераспределять ее объем для хранения отдельных массивов команд, данных и реализации стека в зависимости от решаемых задач. Таким образом, обеспечивается возможность более эффективного использования имеющегося объема оперативной памяти в каждом конкретном случае применения МП. Использование общей шины для передачи команд и данных значительно упрощает отладку, тестирование и текущий контроль функционирования системы, повышает ее надежность. Поэтому Принстонская архитектура в течение долгого времени доминировала в вычислительной технике. Однако ей присущи и существенные недостатки. Основным из них является необходимость последовательной выборки команд и обрабатываемых данных по общей системнойшине. При этом общая шина становится «узким местом», которое ограничивает производительность цифровой системы.

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

### История

- Нулевое поколение — механические компьютеры (1642–1945)
  - Налоговый сумматор (Паскаль), калькулятор на 4 действия (Лейбниц)
- Первое поколение — электронные лампы (1945–1955)
  - COLOSSUS (1943, Тьюринг), ENIAC (1946, Моушли), IAS (1951, фон Нейман)
- Второе поколение — транзисторы (1955–1965)
  - TX-0 (1955, МТИ), PDP-1 (1961, DEC), PDP-8, 7090 (IBM), 6600 (1964, CDC)
- Третье поколение — интегральные схемы (1965–1980)
  - Семейство System/360 (1965, IBM), PDP-11 (1970, DEC)
- Четвертое поколение — сверхбольшие интегральные схемы (1980–?)
  - IBM PC (1981), Apple, Intel, IBM, Dec, Compaq, HP, Sun...
- Пятое поколение — небольшие и «невидимые» компьютеры (1989–?)

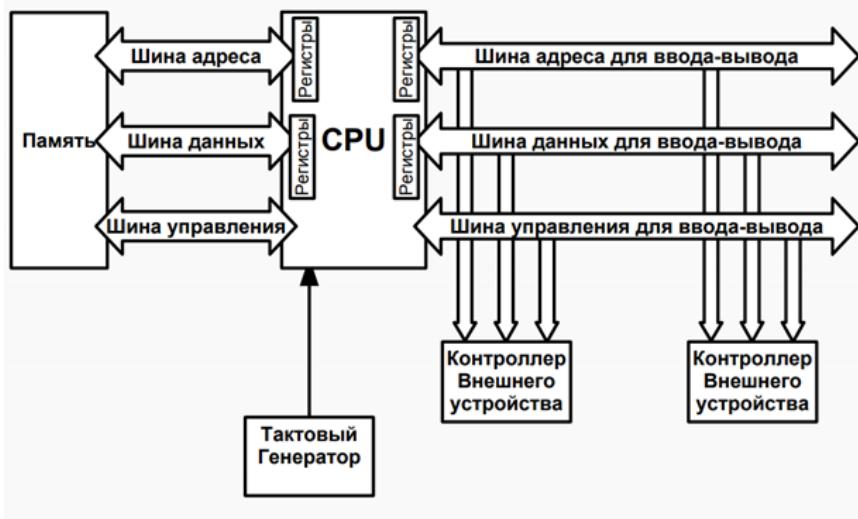
Невидимые – те которые мы не замечаем, по типу каких-то процессоров в стиральных машинах и т.д.



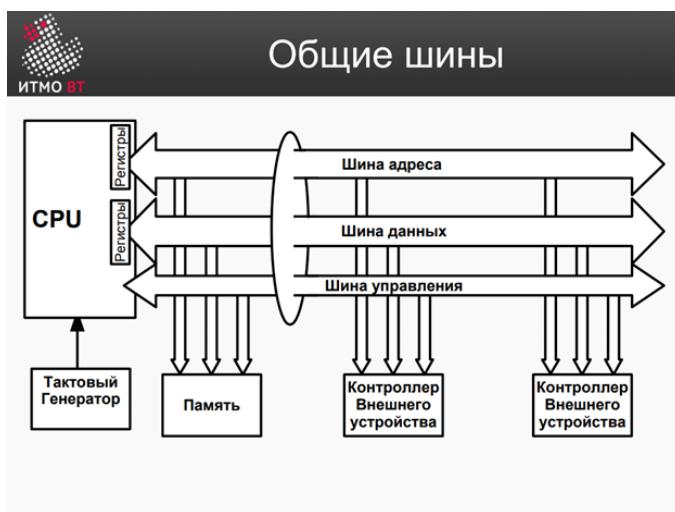
После того, как процессоры стали превосходить по скорости память, перешли к архитектуре, когда процессор является центральным звеном.

Процессор стал быстрее, память стала быстрее, а контроллеры ввода-вывода по скорости остались примерно на том же уровне, поскольку устройства не поменялись.

## Раздельные шины

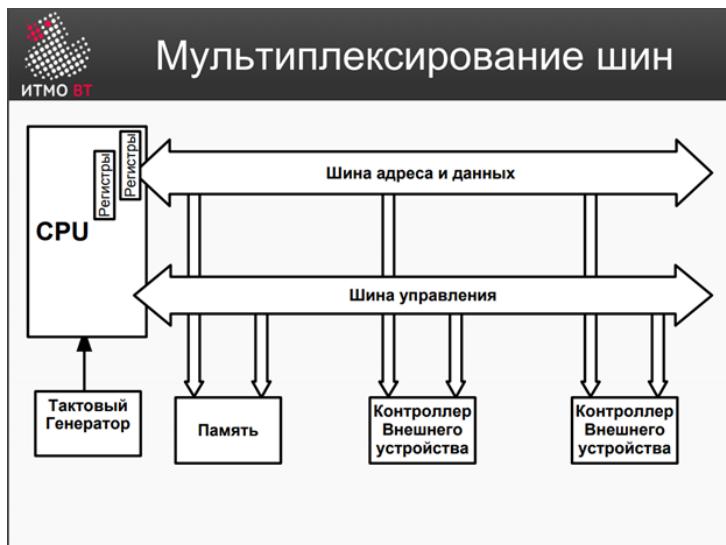


Ну решили дабы сократить объем использовать общую шину и для контроллеров, и для памяти. С памятью мы работаем чаще она быстрее с контроллером реже он медленнее и примерно нет потери производительности

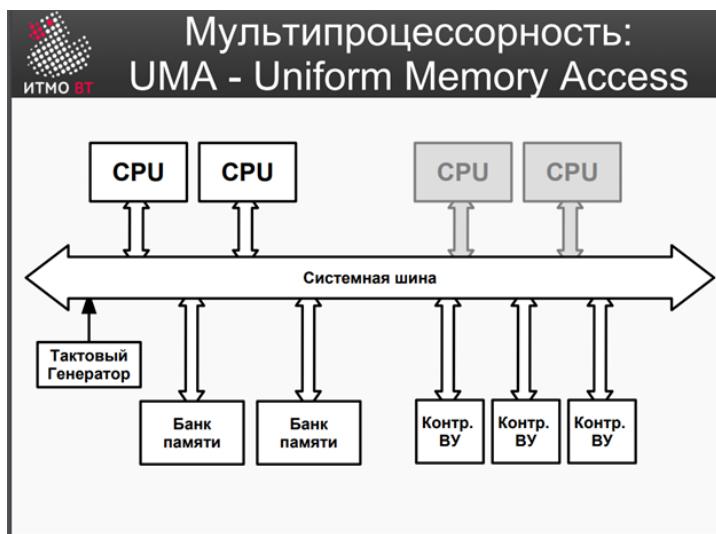


Решили еще упростить, соединив шину адреса и шину данных решив передавать адрес и данные по одним и тем же физическим проводам. В один импульс ТГ предается адрес а в другой данные.

Им не сильно увлекались. Оно, собственно, и понятно



И от этого всего пришли к архитектуре UMA(Uniform Memory Access)



Системная шина – является общим местом для обмена информации между блоками.

Обычно 1 процессор – он самый быстрый, дальше в 1000 раз медленнее это память, и в 1000000 раз медленнее это контроллеры ввода-вывода.

Существует достаточно давно, и не приятным моментом является как раз шина.

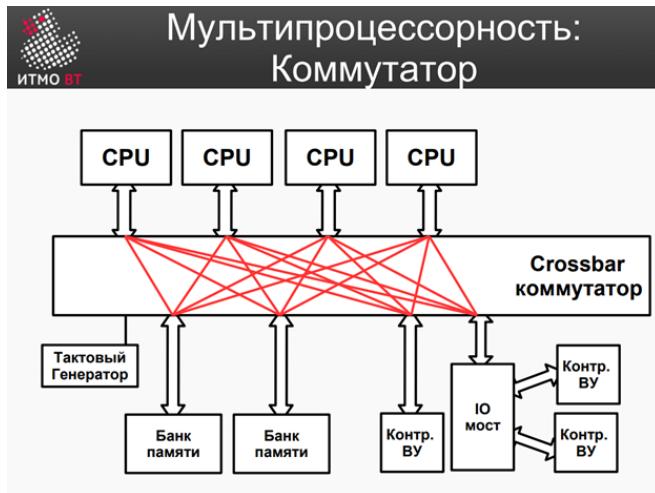
Пока процессор 1, все нормально все функционирует. Проблемы начались тогда, когда люди решили поставить 2й процессор, поскольку уперлись в тактовую частоту, ведь больше 5ГГц физика уже ничего не позволяет. У нас за 1 нс свет проходит 30 см , а за 1 такт  $2/10 * 30 = 6$  см

И решили добавить еще процессоров, и все сломалось, надо стало думать о синхронизации , переключение контекстов и так далее.

Когда нам в магазине говорят, что процессор имеет 2 ядра 4 потока , то потоки – наборы регистров, которые используются для переключения контекстов и сохранения состояния. И это называется гипер трдинг =)))) т.е если хотим выполнять несколько программ на одном процессоре, то у нас есть прерывания от таймера, и в один такт этого таймера выполняется одна программа, а в другой другой. И надо переключать контексты.

Ну так вот люди добавили второй процессор, переписали операционные системы и многие программы.

А вот попытка поставить третий процессор привела к неудаче, машина стала медленнее, все уперлось в системную шину, потому что когда один процессор работает с системной шиной, он является инициатором обмена, он занимает столько сколько ему нужно полос на этойшине и когда мы добавляем второй процессор, получается так , что когда один процессор работает с шиной, другой обрабатывает программу во внутренних регистрах, а когда мы добавляем третий , то шина является узким местом. В ПК существует до сих пор



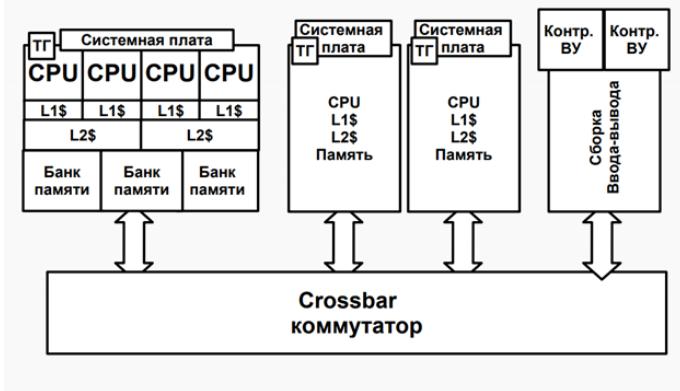
Проблему решил коммутатор. Просто полно связная архитектура, где каждый порт соединен с каждым. Добились, того , что убрали узкое горлышко Т.е каждый процессор в один момент времени может обратится к одному из устройств. Т.е 2 процессора могут обратится к одному банку памяти и они могут это делать параллельно и одновременно. В результате производительность сильно поднялась. Также тут в явном виде разделили устройства на быстродействующие и медленнодействующие, которые соединены с коммутатором через мост.

Высокоскоростная микросхема подключается к порту коммутатора и раздает на все порты процессоров связи.

Uniform Memory Access (сокращённо UMA — «однородный доступ к памяти») — архитектура многопроцессорных компьютеров с 38 общей памятью. Все микропроцессоры в UMA-архитектуре используют физическую память одновременно. При этом время запроса к данным из памяти не зависит ни от того, какой именно процессор обращается к памяти, ни от того, какой именно чип памяти содержит нужные данные.

Далее как дальнейшее развитие этой архитектуры появилась архитектура NUMA

NUMA (Non-Uniform Memory Access — «неравномерный доступ к памяти» или Non-Uniform Memory Architecture — «Архитектура с неравномерной памятью») — схема реализации компьютерной памяти, используемая в мультипроцессорных системах, когда время доступа к памяти определяется её расположением по отношению к процессору



Предназначена для того, чтобы повышать количество процессоров еще больше, может содержать 100+ процессоров.

Каждый процессор организован со своей локальной памятью, с кешем локальным, со своим тактовым генератором, и это все объединено в так называемую системную плату. Системная плата похожа на UMA. Все эти платы соединены между собой коммутатором. В результате все эти процессоры работают под управлением одной операционной системы и вся память, которая находится в системных платах общая. И, собственно, 1 процесс, когда он работает внутри одного ядра занимает определенный процессор и операционная система умная, она старается расположить нужные ему ресурсы в локальной памяти этого процессора. Т.е всегда старается сделать так что бы процессор всегда работал внутри своей локальной платы и если ему нужно больше памяти он может использовать память других системных плат. Таким образом память становится у процессора разной по скорости доступа. Очевидно. Главное чтобы процесс не уходил на другие системные платы, поскольку это приводит к остыванию кэшей. Т.е данные кэшей начинают вытесняться другими процессорами

- + Тактовые генераторы существуют на каждой плате , что позволяет на горячую менять компоненты
- + Процессоры могут теоретически работать на разной тактовой частоте

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



### Современные коммерческие процессоры

- 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

\*Суперскалярный – больше 1 команды за 1 такт тактового генератора.



### CISC, RISC, VLIW

- Complex Instruction Set Computer
  - Традиционные процессоры (например Intel), отягощенные совместимостью
- Reduced Instruction Set Computer
  - Простой набор инструкций, выполнение инструкции за такт
- Very Long Instructions Word
  - Несколько инструкций, упакованных в одну команду
  - Упаковка операций в инструкцию ложится на компилятор

на всех современных системах - системный процессор.

Подведем итоги по примерам:

Структура процессора сформирована так, чтобы последовательно выполнять следующие этапы:

Чтение команд

Преобразование команд (Расшифровка и тд и тп)

Исполнение команд

Таким образом наша БЭВМ сделана подобной процессору или его ядру.

В примерах разобрано и окружение процессора (как устройства).

---

О типах команд:

**CISC** - Complex Instruction Set Computer

- Традиционные процессоры (например Intel), отягощенные совместимостью
- Самостоятельная инструкция выполняет полностью конкретную задачу (например инструкция логического или - два хода)

**RISC** - Reduced Instruction Set Computer

- Простой набор инструкций, выполнение инструкции за такт
- Кусочек выполнения задачи, подобен микрокоманде или какому-то объединению - загрузка из памяти.

**VLIW** - Very Long Instructions Word

- Несколько инструкций, упакованных в одну команду
- Упаковка операций в инструкцию ложится на компилятор
- Как для RISC - CISC, так для CISC - VLIW

Intel использует cisc и это историческая архитектура в которой много мусора, который хранится ради поддержки программ на прошлых сборках

Как показала практика разница в производительности между RISC и CISC небольшая поэтому интел так и остался CISC

**CISC** - Концепция проектирования процессоров, которая характеризуется следующим набором свойств: · большим числом различных по формату и длине команд; · введением большого числа различных режимов адресации; · 40 · обладает сложной кодировкой инструкции. Процессору с архитектурой CISC приходится иметь дело с более сложными инструкциями неодинаковой длины. Выполнение одиночной CISC-инструкции может происходить быстрее, однако обрабатывать несколько таких инструкций параллельно сложнее. Облегчение отладки программ на ассемблере влечет за собой загромождение узлами микропроцессорного блока. Для повышения быстродействия следует увеличить тактовую частоту и степень интеграции, что вызывает необходимость совершенствования технологии и, как следствие, более дорогостоящего производства. Для CISC-архитектуры типичны:

- наличие в процессоре сравнительно небольшого числа регистров общего назначения;
- большое количество машинных команд, некоторые из них аппаратно реализуют сложные операторы ЯВУ;
- разнообразие способов адресации operandов;
- множество форматов команд различной разрядности;
- наличие команд, где обработка совмещается С обращением к памяти.

**RISC**

Процессор с сокращенным набором команд. Система команд имеет упрощенный вид. Все команды одинакового формата с простой кодировкой. Обращение к памяти происходит посредством команд загрузки и записи, остальные команды типа регистр-регистр. Команда, поступающая в CPU, уже разделена по полям и не требует дополнительной дешифрации. Команда меньше загромождает ОЗУ, CPU дешевле. Программной совместимостью указанные архитектуры не обладают. Отладка программ на RISC более сложна. Поскольку RISC-инструкции просты, для их выполнения нужно меньше логических элементов, что в конечном итоге снижает стоимость процессора. Но большая часть программного обеспечения сегодня написана и откомпилирована специально для CISC-процессоров фирмы Intel. Для использования архитектуры RISC нынешние программы должны быть перекомпилированы, а иногда и переписаны заново.

**VLIW** - архитектура процессоров с несколькими вычислительными устройствами. Характеризуется тем, что одна инструкция процессора содержит несколько операций, которые должны выполняться параллельно, базируется на RISC-архитектуре. – Несколько инструкций, упакованных в одну команду – Упаковка операций в инструкцию ложится на компилятор. 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 как одна сверхдлинная команда.

| Характеристика               | CISC                                               | RISC                                      | VLIW                                        |
|------------------------------|----------------------------------------------------|-------------------------------------------|---------------------------------------------|
| Длина команды                | Варьируется                                        | Единая                                    | Единая                                      |
| Расположение полей в команде | Варьируется                                        | Неизменное                                | Неизменное                                  |
| Количество регистров         | Несколько (часто специализированных)               | Много регистров общего назначения         | Много регистров общего назначения           |
| Доступ к памяти              | Может выполняться как часть команд различных типов | Выполняется только специальными командами | Выполняется только специальными командами – |

### Структура современного процессора

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

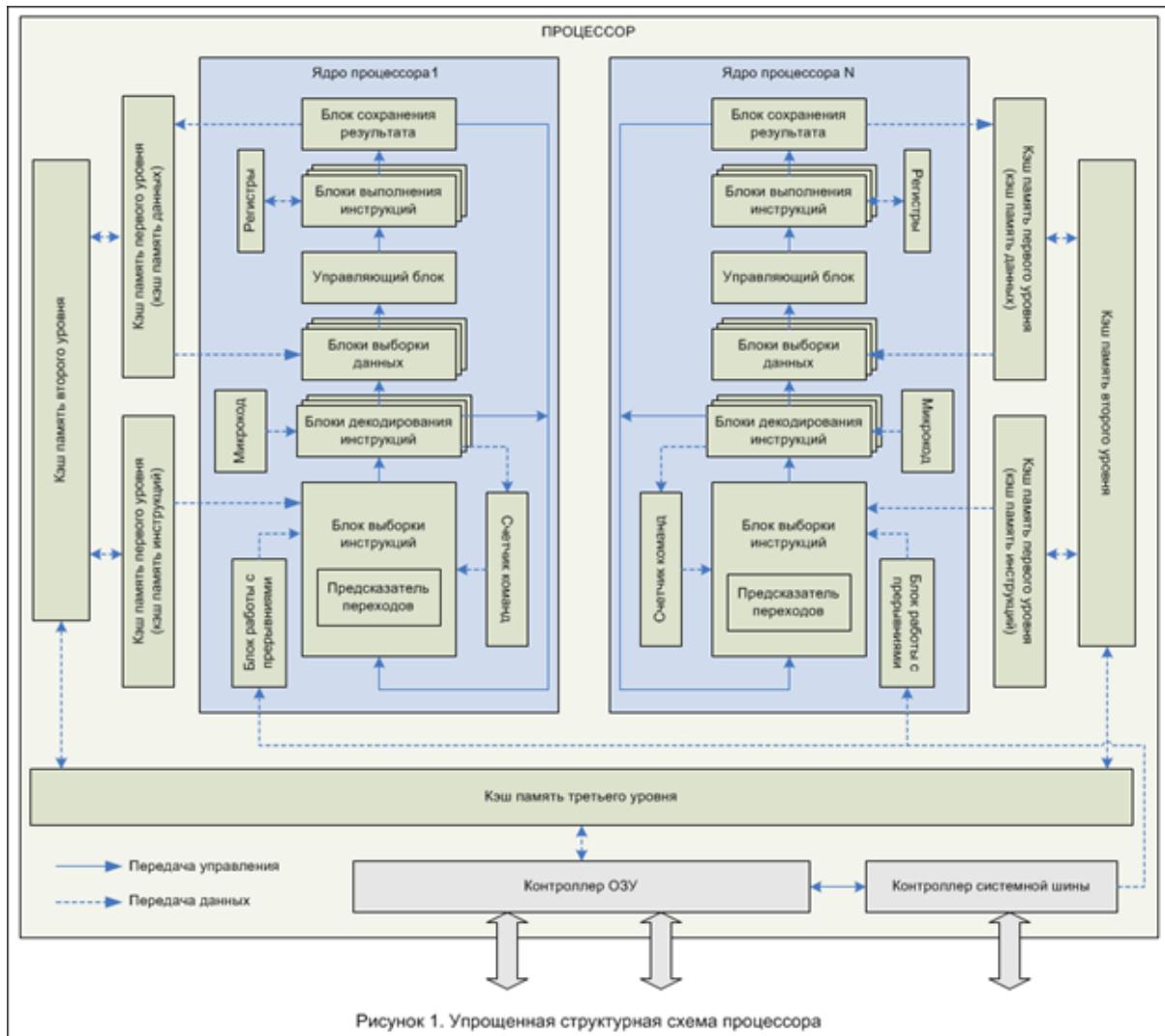


Рисунок 1. Упрощенная структурная схема процессора

Принцип работы ядра процессора основан на цикле, описанном еще Джоном фон Нейманом в 1946 году. В упрощенном виде этапы цикла работы ядра процессора можно представить следующим образом:

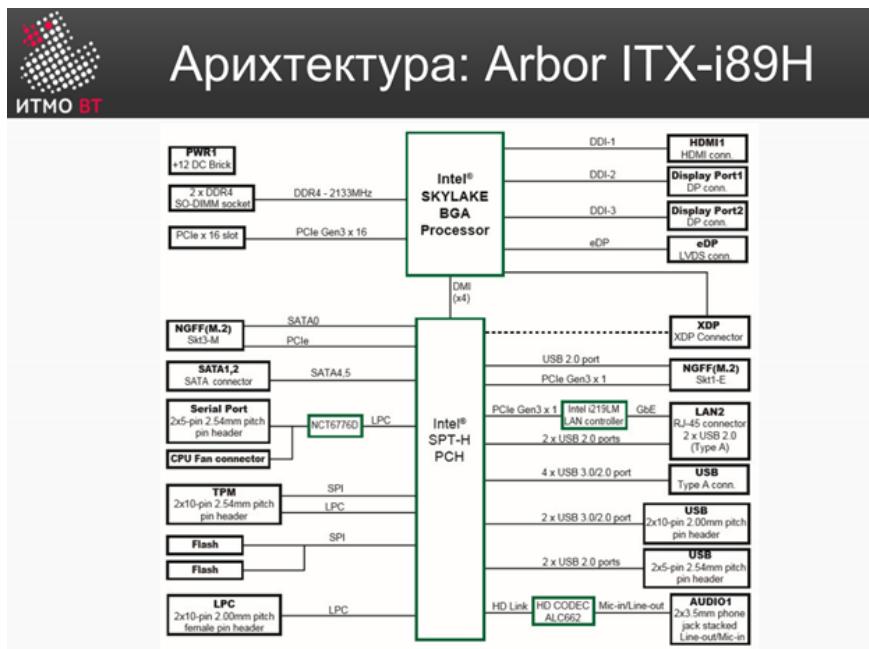
1. Блок выборки инструкций проверяет наличие прерываний. Если прерывание есть, то данные регистров и счетчика команд заносятся в стек, а в счетчик команд заносится адрес команды обработчика прерываний. По окончанию работы функции обработки прерываний, данные из стека будут восстановлены;
2. Блок выборки инструкций из счетчика команд считывает адрес команды, предназначенный для выполнения. По этому адресу из КЭШ-памяти или ОЗУ считывается команда. Полученные данные передаются в блок декодирования;
3. Блок декодирования команд расшифровывает команду, при необходимости используя для интерпретации команды записанный в ПЗУ микрокод. Если это команда перехода, то в счетчик команд записывается адрес перехода и управление передается в блок выборки инструкций (пункт 1), иначе счетчик команд увеличивается на размер команды (для процессора с длинной команды 32 бита – на 4) и передает управление в блок выборки данных;
4. Блок выборки данных считывает из КЭШ-памяти или ОЗУ требуемые для выполнения команды данные и передает управление планировщику;

5. Управляющий блок определяет, какому блоку выполнения инструкций обработать текущую задачу, и передает управление этому блоку;
6. Блоки выполнения инструкций выполняют требуемые командой действия и передают управление блоку сохранения результатов;
7. При необходимости сохранения результатов в ОЗУ, блок сохранения результатов выполняет требуемые для этого действия и передает управление блоку выборки инструкций (пункт 1).

Описанный выше цикл называется процессом (именно поэтому процессор называется процессором). Последовательность выполняемых команд называется программой.

Скорость перехода от одного этапа цикла к другому определяется тактовой частотой процессора, а время работы каждого этапа цикла и время, затрачиваемое на полное выполнение одной инструкции, определяется устройством ядра процессора.

### **SPARC – много ядер но само ядро не такое мощное как у интел**



Микросхемы поддержки или чипсет(интел)

У процессора только в основном только высокоскоростные шины. Шина для обращения к памяти.

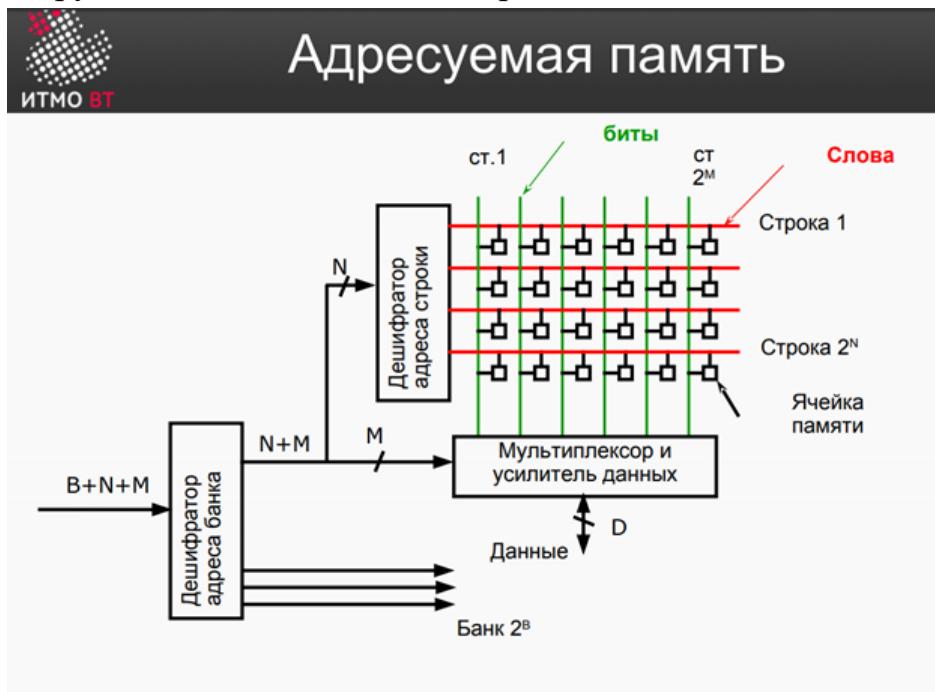
Есть несколько слотов по типу PCI

Высоко производительная графика порты HDMI

Все то, что необходимо дальше такое как контроллер внешних дисков, изернет контроллер все они управляются микросхемой поддержки, которая связана с процессором каналом прямого доступа к памяти



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



Мультиплексор – осуществляет передачу с входов на выход неизменно, однако входов у него много, а выход 1

Он просто передает данные с выбранного столбца на выход

Современная память обычно разбивается на банки



**15432** @15432

Системный программист ^\_^

- ✓ На полке в кладовке стоят банки с огурцами. Чтобы достать конкретный один огурец, надо сначала достать с полки банку и открыть её. Почему в кладовке столько банок? Почему бы не сделать банку размером с кладовку? А вот дорого такую большую ёмкость из стекла ваять, да и нырять за огурцами придётся, сложно будет найти нужный среди всей массы. А так и банки небольшие, в руках удержать можно, и подписаны все - в каком году и месяце закатка была.
- Ответ написан более года назад

[Нравится](#)

[Комментировать](#)



На вход подается адрес в виде последовательности сгруппированных битов

| Bank | N | M |
|------|---|---|
|      |   |   |

И каждый банк представлен такой матрицей

Выбирается строка и столбец и их пересечение с помощью мультиплексора попадает на выход

Т.е грубо говоря вся строка поступает на входы мультиплексора но M указывает на нужный элемент.

Временные диаграммы

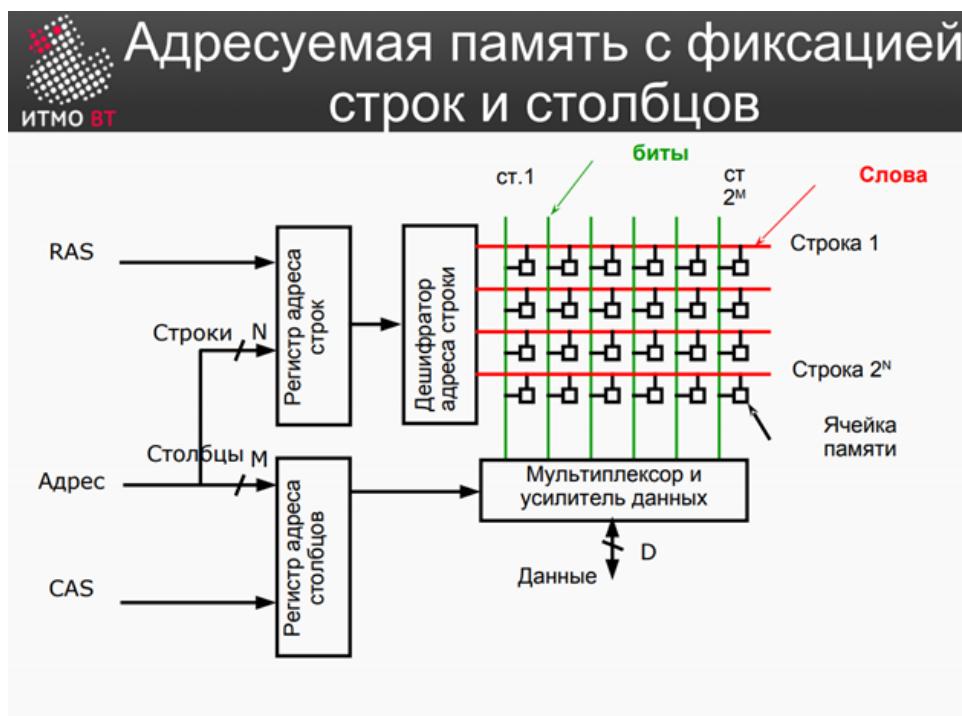


Все очевидно, однако важно отметить, что линия W/R – не на шине, а внутри памяти.

По сигналу чтения начинают срабатывать дешифраторы

Тд – время доступа

Тв – время восстановления



Предыдущие слайды сложно сделать в больших масштабах.

Похоже на мультиплексирование

Адрес передавать не целиком а разбитый на 2 части: адрес строки и адрес столбца и будем передавать их последовательно по одной шине

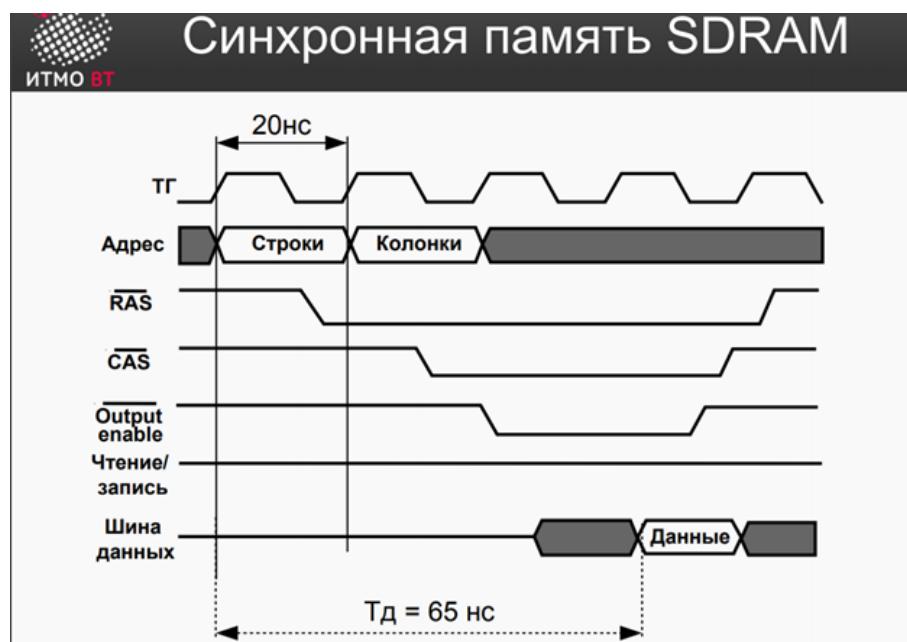
Банка не изменилась =)

Появились 2 дополнительных регистра, как показано на рисунке и 2 дополнительных сигнала RAS(Address strobe) и CAS (RAW и COLUMN ) т.е выбор строки и выбор столбца

В результате половина адреса передается в один такт, половина в другой.

А дальше все как всегда

Память стала чуть медленнее но не в 2 раза а менее, за счет регистров которые можно инкрементить



Это уже то что работает в нашем ПК

Есть ТГ отсюда и синхронность

Output enable – когда память читает

1. Передали адрес строки подтвердив его RAS
2. Передали адрес столбца подтвердив его CAS
3. Подключили выход памяти(Output enable)
4. Далее начинают появляться данные
5. Возврат в исходное состояние



## 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
- ....

Burst mode – режим, при котором на запрос по конкретному адресу возвращаются не только данные, хранящиеся по этому адресу, но и пакет данных по нескольким последующим адресам.

SPD – определяет память там прописано на какой частоте она должна работать его читает процессор.

Расслоение памяти- расположение кусков памяти, которое позволяет параллельно читать больше данных

При расслоении памяти соседние по адресам ячейки размещаются в различных модулях памяти, так что появляется возможность производить несколько обращений одновременно. Например, при расслоении на два направления ячейки с нечетными адресами оказываются в одном модуле памяти, а с четными — в другом. При простых последовательных обращениях к основной памяти ячейки выбираются поочередно.

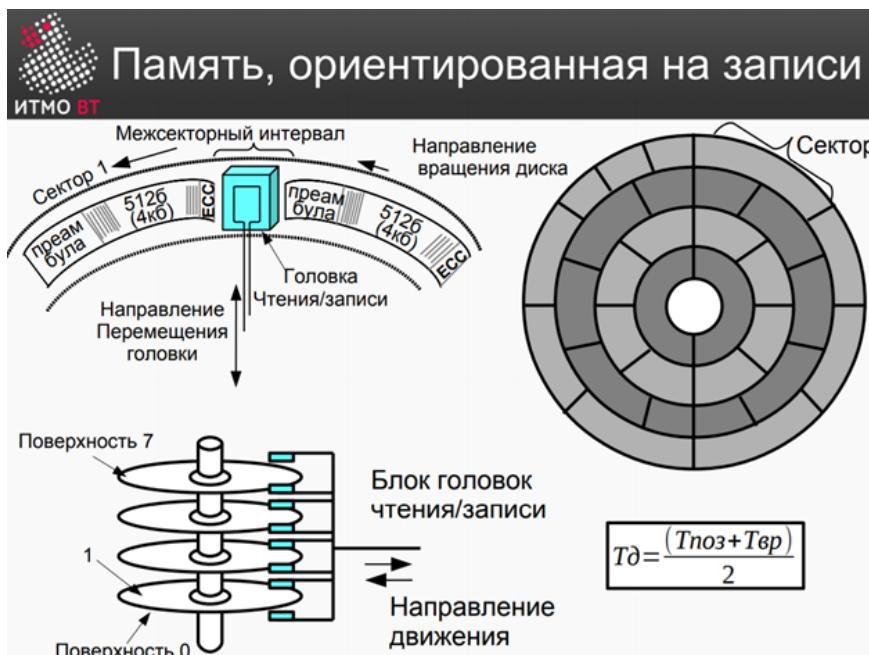


Рисунок 2.3.13 – расслоение памяти по четырем банкам.

17000 во столько раз быстрее эталон

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

Блочная структура (схема). Адресное пространство памяти разбивается на группы последовательных адресов, и каждая такая группа обеспечивается отдельным банком памяти



#### Устройство жестких дисков

В них есть блины, обозначает поверхности, на которых расположена информация.

\*История о том, как КСВ раскручивал диск с толкача\*

Каждый блин содержит верхнюю поверхность и нижнюю.

Кроме того, есть некоторое количество головок чтения/записи.

Принцип работы заключается в том, что головка “идет” по поверхности диска, которая разбита на сектора и включает в себя:

1. Преамбулу для настройки- позволяет, так сказать, синхронизироваться перед чтением или записью

## 2. Данные

3. ECC (Error Correction Code)- используется код Хэмминга или чаще код Рида-Соломона, позволяющий исправлять множественные ошибки, а не только одиночные. Между соседними секторами находится межсекторный интервал. Можно назвать контролем четности

Физическая структура самого диска – серая картинка на слайде

Дорожка в форме круга и разбита на сектора, кроме того, все дорожки имеют разный радиус

Время доступа к диску определяется по той самой формуле на слайде. Пояснение к происхождению формулы:

Т позиционирования (двигаем головку) + Т поворота до нужного блока(крутим диск)

Далее берем среднее поскольку каждый раз блок находится на разном расстояние от текущего положения головки.

Также мы помним, что все дорожки имеют разный радиус, тогда их угловая скорость разная. И получается что чем дальше от центра вращения тем больше скорость записи и чтения(обычно между граничными дорожками разница составляет 40%) РАЗМЕЩАЕМ ОБЛАСТЬ ПОДКАЧКИ В НАЧАЛЕ ДИСКА, ЧТОБЫ КОМПУКТЕР БЫСТРЕЕ РАБОТАЛ И СОЛНЦЕ СВЕТИЛО ЯРЧЕ.

Диски – память с блочным доступом и каждый раз передается блок диска

Бывает случайный доступ и последовательный, разница в скорости доступа в 30-40 раз



Ленточные накопители существуют даже в наши дни, поскольку их главным плюсом являются очень дешевая стоимость хранения

Стоит помнить, что чем быстрее лента протягивается относительно головки тем быстрее происходит взаимодействие. Проблема в том, что максимальная скорость движения ленты ограничена. В результате решили вращать головку, а не ленту

В современных ленточных накопителях головку наклонили относительно ленты, которая медленно двигается и в результате получается рисунок дорожек как на слайде.

И соответственно крутить головку намного проще, и это дает большой прирост в скорости чтения/записи

Вернемся к понятию скорости доступа, и очевидно, что в таких накопителях она очень большая

Поэтому лента используется в качестве хранения для бэкапов, которые целиком пишутся на нее

## 28. Характеристики запоминающих устройств. Пирамида памяти



### Характеристики памяти

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

Чем ближе к процессору, тем более дорогая

Единицы пересылки, то что пересылается



### Характеристики памяти

- Быстродействие и временные соотношения
  - Время доступа Тд
  - Длительность цикла памяти (время обращения) Тц
  - Время чтения и время записи
  - Время восстановления Тв
  - Скорость передачи информации
- Физический тип и особенности
- Стоимость

Время доступа – время от обращения до передачи информации

Цикл памяти – обратились получили, произошло восстановление в исходное состояние

Время восстановления – привести память в исходное состояние

Пример с блоком ленточным, по типу у нас же сдвинулось все и ленту надо мотать обратно

Бод – скорость передачи информации БИТ В БЛЯДСКУЮ СЕКУНДУ

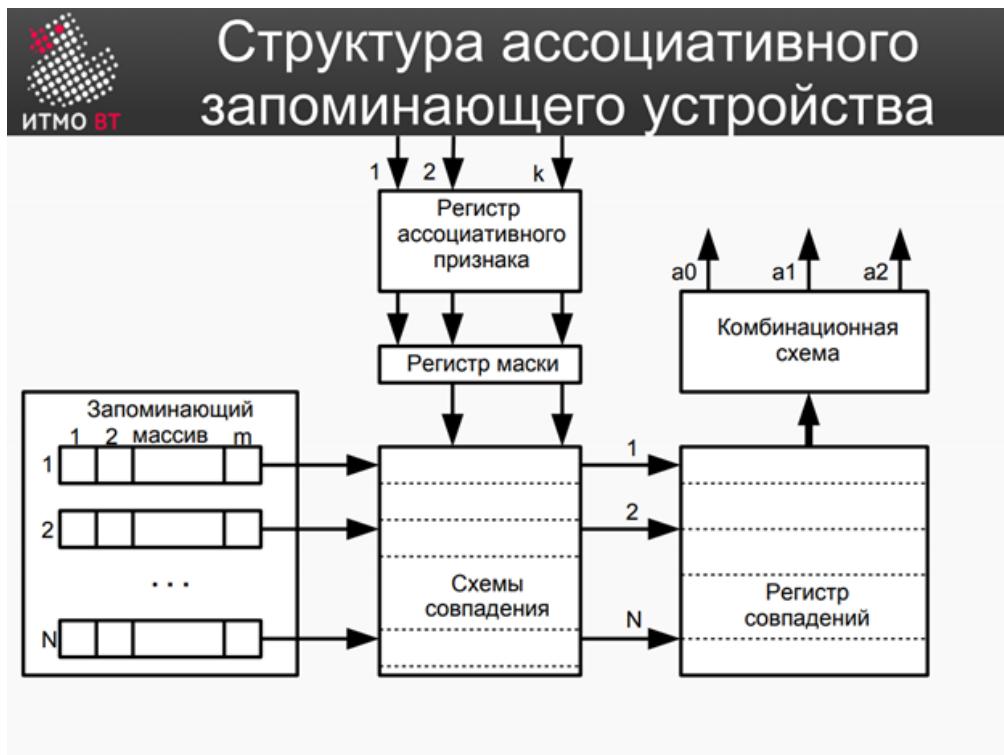
Стоимость – стоимость одного байта



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

\*-ответ девушки на предложение

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



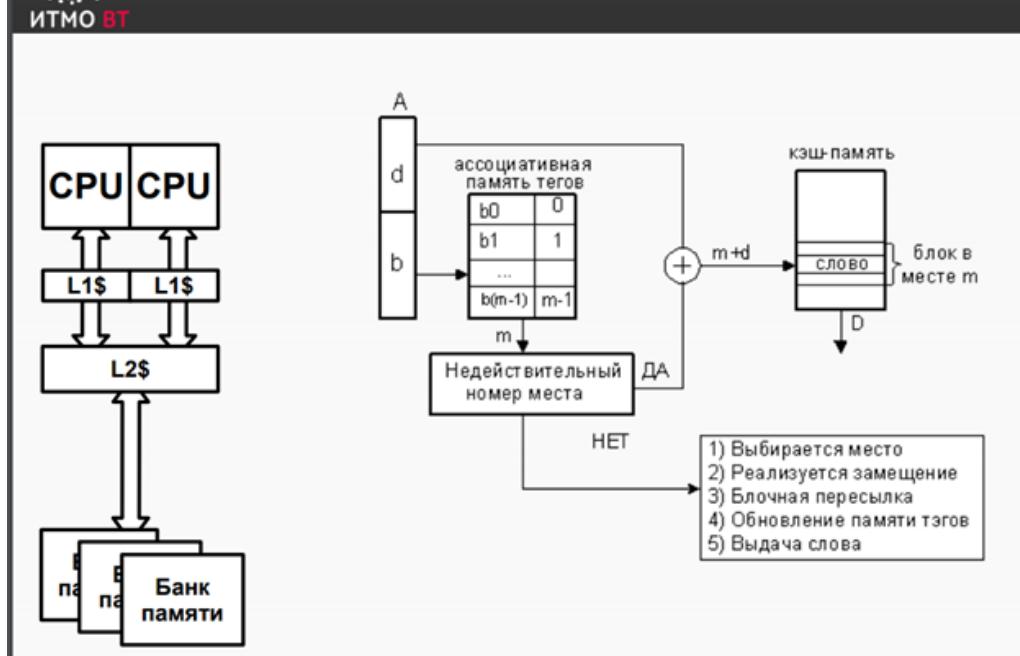
Ассоциативная память построена по другому принципу, она выбирает не информацию по адресу, а по нужному для нас признаку. Т.е у нас есть регистр ассоциативного признака, а каждая ячейка памяти содержит в себе схему сравнения, т.е все что поступает на нее сравнивается на наличие необходимого содержимого и те ячейки которые совпадают передаются при помощи регистров сравнения в дальнейшую комбинационную схему.

Память такая дорогая за счет компараторов

Подобным образом устроен КЭШ

Кэш-память — это высокоскоростная память произвольного доступа, используемая процессором компьютера для временного хранения информации. Она увеличивает производительность, поскольку хранит наиболее часто используемые данные и команды «ближе» к процессору, откуда их можно быстрее получить

# Кэш память



## Структура

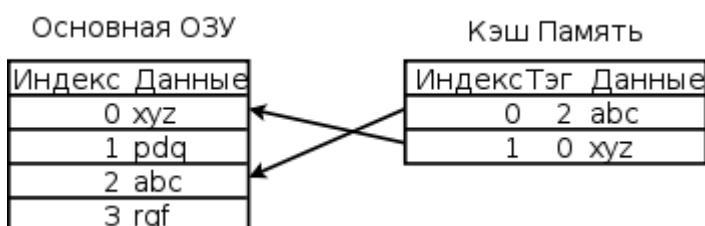
Адрес бьется на некоторое количество частей, которые делятся на так называемый тег и просто адрес

В КЭШ памяти выбирается по тег содержимое и потом выдается строка из кэш памяти

Типичная структура записи в кэше



## Принцип работы из википедии



Каждая строка — группа ячеек памяти содержит данные, организованные в кэш-линии. Размер каждой кэш-линии может различаться в разных процессорах, но для большинства x86-процессоров он составляет 64 байта. Размер кэш-линии обычно больше размера данных, к которому возможен доступ из одной машинной команды (типовыи размеры от 1 до 16 байт). Каждая группа данных в памяти размером в 1 кэш-линию имеет порядковый номер. Для основной памяти этот номер является адресом памяти с отброшенными

младшими битами. В кэше каждой кэш-линии дополнительно ставится в соответствие тег, который является адресом продублированных в этой кэш-линии данных в основной памяти.

При доступе процессора в память сначала производится проверка, хранит ли кэш запрашиваемые из памяти данные. Для этого производится сравнение адреса запроса со значениями всех тегов кэша, в которых эти данные могут храниться.



Как мы знаем процессор обращается сначала к КЭШу и поскольку это ассоциативный тип памяти, если ничего не обнаружено, то происходит обращение к основной памяти, которая гораздо медленнее

Промах – когда нужного слова не оказалось в КЭШ памяти

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

Виртуальная память — метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем.

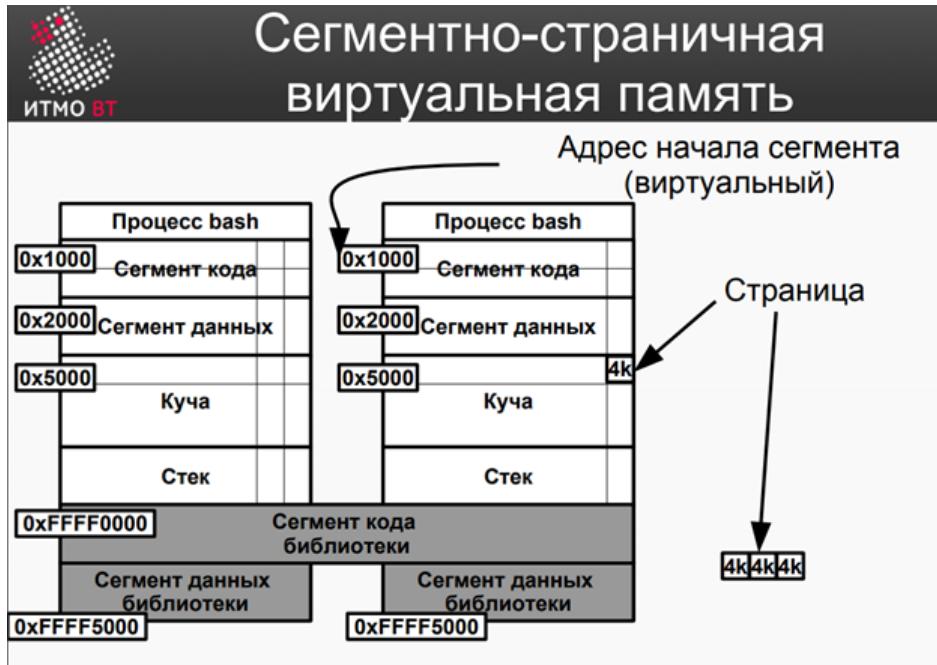
### Страницчная организация виртуальной памяти

В большинстве современных операционных систем виртуальная память организуется с помощью страницочной адресации. Оперативная память делится на страницы: области памяти фиксированной длины, которые являются минимальной единицей выделяемой памяти. Исполняемый процессором пользовательский поток обращается к памяти с помощью адреса виртуальной памяти, который делится на номер страницы и смещение внутри страницы. Процессор преобразует номер виртуальной страницы в адрес соответствующей ей физической страницы при помощи буфера ассоциативной трансляции (TLB). Если ему не удалось это сделать, то требуется дозаполнение буфера путём обращения к таблице страниц, что может сделать либо сам процессор, либо операционная система. Если страница была выгружена из оперативной памяти, то операционная система подкачивает страницу с жёсткого диска в ходе обработки события. При запросе на выделение памяти операционная система может «сбросить» на жёсткий диск страницу, к которым давно не было обращений. Критические данные (например, код запущенных и работающих программ, код и память ядра системы) обычно находятся в оперативной памяти (исключения существуют, однако они не касаются тех частей, которые отвечают за обработку аппаратных прерываний, работу с таблицей страниц и использование файла подкачки).

### Сегментная организация виртуальной памяти

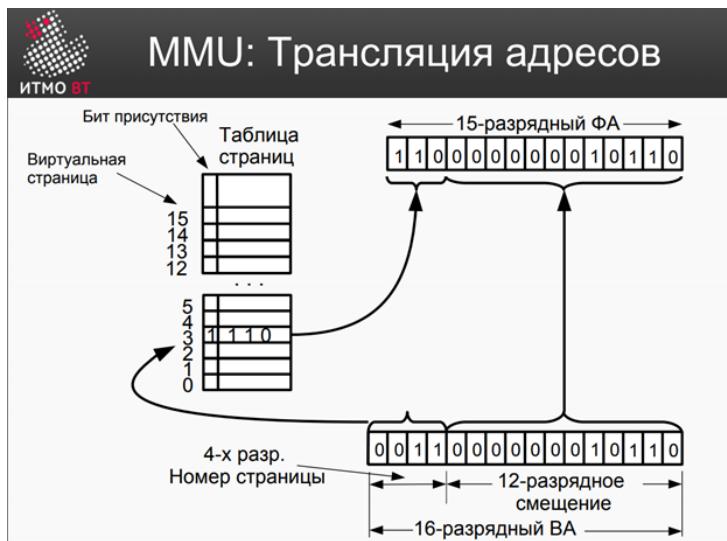
Механизм организации виртуальной памяти, при котором виртуальное пространство делится на части произвольного размера — сегменты. Этот механизм позволяет, к примеру, разбить данные процесса на логические блоки. Для каждого сегмента, как и для страницы, могут быть назначены права доступа к нему пользователя и его процессов. При загрузке процесса часть сегментов помещается в оперативную память, а часть сегментов размещается в дисковой памяти. Во время загрузки система создает таблицу сегментов процесса, в которой для каждого сегмента указывается начальный физический адрес сегмента в оперативной памяти, размер сегмента, правила доступа, признак модификации, признак обращения к данному сегменту за последний интервал времени и некоторая другая информация. Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок оперативной памяти, в который данный сегмент загружается в единственном экземпляре. Система с сегментной организацией функционирует аналогично системе со страницной организацией: время от времени происходят прерывания, связанные с отсутствием нужных сегментов в памяти, при необходимости освобождения памяти некоторые сегменты выгружаются, при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Кроме того, при обращении к памяти проверяется, разрешен ли доступ требуемого типа к данному сегменту. Виртуальный адрес при сегментной организации памяти может быть представлен парой ( $g, s$ ), где  $g$  — номер сегмента, а  $s$  — смещение в сегменте. Физический адрес получается путём сложения начального физического адреса сегмента, найденного в таблице сегментов по номеру  $g$ , и смещения  $s$ . Недостатком данного метода распределения памяти является фрагментация на уровне

сегментов и более медленное по сравнению со страничной организацией преобразование адреса.



Каждый процесс в современной вычислительной машине разделен на сегменты и каждый сегмент находится в адресуемой памяти на своем месте, причем важно обратить внимание на то, что в виртуальном адресном пространстве все адреса одинаковые, что видно на слайде. Важно то, что внутри процессов адреса одинаковые, но они ссылаются на разные физические адреса памяти

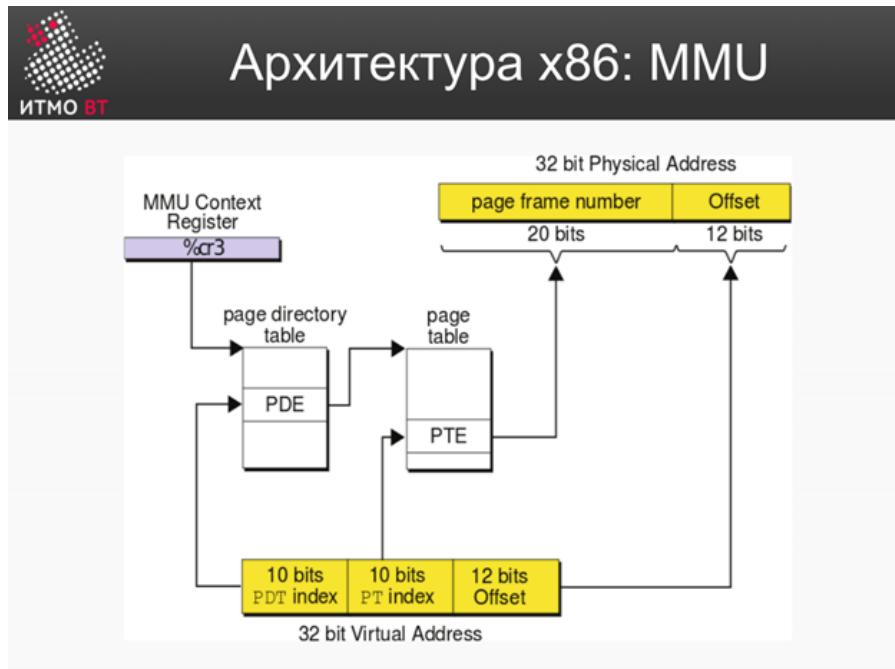
Также существует такое понятие, как разделяемые библиотеки, которые могут подключаться за счет виртуальной памяти сразу в несколько процессов



MMU – Memory Management Unit

У него на вход поступает виртуальный адрес, который с помощью таблицы преобразование проверяет наличие страницы в виртуальной памяти и какой у нее физический адрес

Этим занимает операционная система



2 обращения к таблицам



а тут 4 =(



## TLB

- Кэширует часто используемые преобразования
- Обычно раздельный для адреса и данных
- Организован в виде ассоциативной памяти

Записываем туда виртуальные адреса и соответствующие им физические, тем самым избегаем частого обращения к памяти

Когда виртуальный адрес необходимо преобразовать в физический адрес, то вначале просматривается TLB. Если совпадение найдено (называется «TLB hit») то возвращается физический адрес, и доступ к памяти может продолжаться. Однако, если нет совпадения (называемого «TLB miss») то либо модуль управления памятью либо обработчик пропусков TLB операционной системы ищет сопоставление адресов в таблице страниц, чтобы узнать, существует ли сопоставление (называется «page walk»). Если сопоставление существует — то оно записывается обратно в TLB (это необходимо сделать, поскольку аппаратное обеспечение обращается к памяти через TLB в системе виртуальной памяти), и текущая команда перезапускается (что также может происходить параллельно).

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

Под сетью ЭВМ понимают соединение двух и более ЭВМ с целью совместного использования их ресурсов (процессоров, устройств памяти, устройств ввода/вывода, данных). По степени охвата территории различают сети:

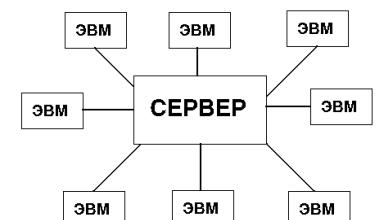
- локальные (местные) - в пределах одного учреждения, помещения (или при максимальном удалении ЭВМ не более 1км.)
- региональные - внутри населенного пункта, района
- национальные - внутри государства
- глобальные

По степени доступности различают корпоративные и общедоступные сети.

По топологии:

- звездаобразная топология

При таком способе обмен данными между ЭВМ осуществляется через более мощную ЭВМ - сервер. Недостатком такого соединения является низкая живучесть сети - выход из строя сервера означает прекращение функционирования сети. Однако, простота и дешевизна реализации сделала эту структуру популярной в локальных сетях.



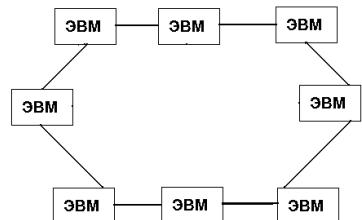
- топология с общей шиной

При этом способе обмен данными происходит через общую шину, которую использует механизм прерывания может "захватывать" тот или иной компьютер. Характерной особенностью здесь является отсутствие сервера. Очень часто используется в локальных сетях, а уж в "домашних" повсеместно.



- кольцевая топология

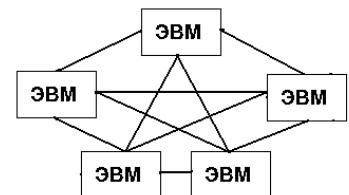
В этой структуре каждая ЭВМ использует механизм прерывания работает в качестве ретранслятора. Обратите внимание, живучесть сети повышена - при одиночном обрыве связи между соседними ЭВМ сеть продолжает функционировать



- полная топология

Соединение ЭВМ "каждая с каждой" позволяют получить сеть самую дорогую, но и обладающую максимальной живучестью.

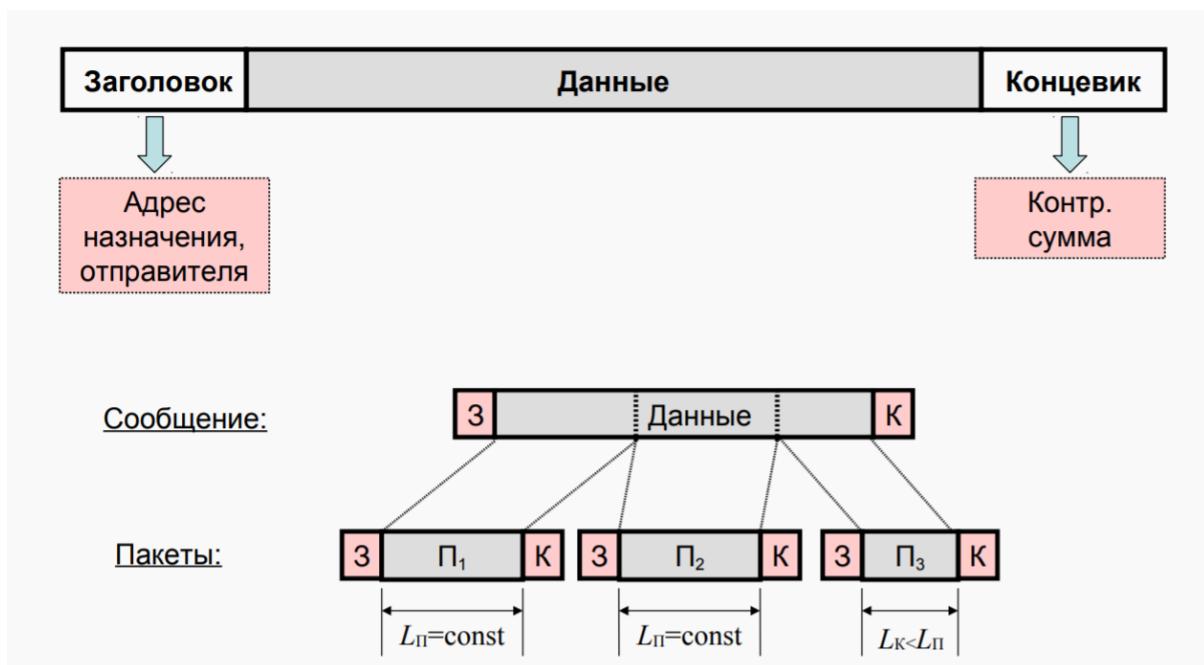
Сообщение — цифровые данные определенного формата, предназначенные для передачи.



Пакет — это определенным образом оформленный блок данных, передаваемый по сети. Часто состоит из заголовка и полезной нагрузки.

| Уровень (layer) | Тип данных (PDU) | Функции | Примеры                           |
|-----------------|------------------|---------|-----------------------------------|
| Host layers     | 7. Прикладной    | Данные  | Доступ к сетевым службам          |
|                 | 6. Представления |         | Представление и шифрование данных |
|                 | 5. Сеансовый     |         | Управление сеансом связи          |

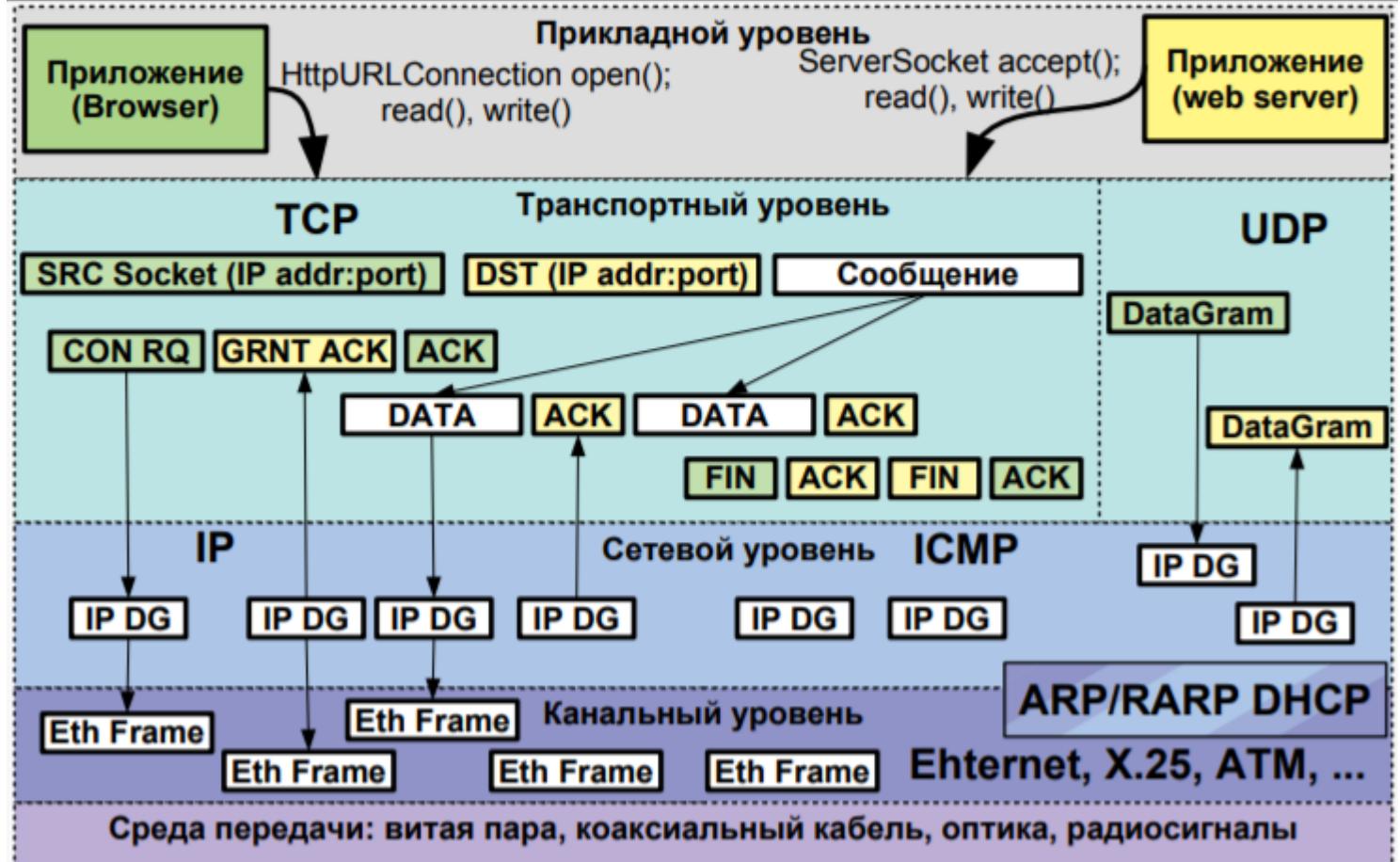
|              |                 |                       |                                                          |                                                                      |
|--------------|-----------------|-----------------------|----------------------------------------------------------|----------------------------------------------------------------------|
|              | 4. Транспортный | Сегменты /Дейтаграммы | Прямая связь между конечными пунктами и надёжность       | TCP, UDP, SCTP, PORTS                                                |
| Media layers | 3. Сетевой      | Пакеты                | Определение маршрута и логическая адресация              | IPv4, IPv6, IPsec, AppleTalk                                         |
|              | 2. Канальный    | Биты/Кадры (frame)    | Физическая адресация                                     | PPP, IEEE 802.22, Ethernet, DSL, ARP, сетевая карта.                 |
|              | 1. Физический   | Биты                  | Работа со средой передачи, сигналами и двоичными данными | кабель (USB, «витая пара», коаксиальный, оптоволоконный), радиоканал |



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

## Модель TCP/IP

ИТМО ВТ



**TCP/IP** — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи). Наборы правил, решающих задачу по передаче данных, составляют стек протоколов передачи данных, на которых базируется Интернет. Название TCP/IP происходит из двух важнейших протоколов семейства — Transmission Control Protocol (TCP) и Internet Protocol (IP), которые были первыми разработаны и описаны в данном стандарте.

## Уровни стека TCP/IP [ править | править код ]

Стек протоколов TCP/IP включает в себя четыре уровня<sup>[5]</sup>:

- Прикладной уровень (Application Layer),
- Транспортный уровень (Transport Layer),
- Межсетевой уровень (Сетевой уровень<sup>[6]</sup>) (Internet Layer),
- Канальный уровень (Network Access Layer).

**Канальный уровень** (англ. *Data Link layer*), также **уровень передачи данных**<sup>[1]</sup> — второй уровень сетевой модели OSI, предназначенный для передачи данных узлам, находящимся в том же сегменте локальной сети. Также может использоваться для обнаружения и, возможно, исправления ошибок, возникших на физическом уровне. Примерами протоколов, работающих на канальном уровне, являются: Ethernet для локальных сетей (многоузловой), Point-to-Point Protocol (PPP), HDLC и ADCCP для подключений точка-точка (двухузловой).

Канальный уровень отвечает за доставку **кадров** (*frame*) между устройствами, подключенными к одному сетевому сегменту. Кадры канального уровня не пересекают границ сетевого сегмента. Кадры передаются последовательно с обработкой **кадров подтверждения**, отсылаемых обратно получателем<sup>[1]</sup>.

Функции межсетевой **маршрутизации** и глобальной **адресации** осуществляются на более высоких уровнях модели OSI, что позволяет протоколам канального уровня сосредоточиться на локальной доставке и адресации.

Заголовок кадра содержит **аппаратные адреса** отправителя и получателя, что позволяет определить, какое устройство отправило кадр и какое устройство должно получить и обработать его. В отличие от иерархических и маршрутизуемых адресов, аппаратные адреса одноуровневые. Это означает, что никакая часть адреса не может указывать на принадлежность к какой-либо логической или физической группе.

**Сетевой уровень** (англ. *Network layer*) — 3-й уровень сетевой модели OSI, предназначается для определения пути передачи данных. Отвечает за трансляцию **логических адресов** и имён в **физические**, определение кратчайших маршрутов, **коммутацию** и **маршрутизацию**, отслеживание неполадок и заторов в сети. На этом уровне работает такое сетевое устройство, как **маршрутизатор**.

В пределах семантики иерархического представления модели OSI Сетевой уровень отвечает на запросы обслуживания от **Транспортного уровня** и направляет запросы обслуживания на **Канальный уровень**.

Максимальная длина пакета сетевого уровня может быть ограничена командой **ip mtu**.

### Передающая среда

**Коаксиальный кабель** (устарел) – «толстый» - 10Base-5 — до 500м – «тонкий» - 10Base-2 — до 50м

**Витая пара** 10Base-T, 100Base-T, .... – Категория 3: от 10 до 100 Мбит/с 100BASE-T4 (100м). – Категория 5e: 100 Мбит/с (2 пары), 1Гбит/с на (4пары) – Категория 6: 10 Гбит/с (55м) – Категория 7a: 40Гбит/с (50м), 100Гбит/с (15м)

**Оптика** (10BASE-F,100BASE-SX,10GBASE-ER...) – ST (Straight Tip) – SC (Standard Connector) – LC (Lucent Connector) – Лазер находится в SFP (Small Plugin Factor) – ~500 м (Multi-mode fiber), ~80км (Single Mode)

**Wireless** (802.11 - WiFi, 802.16 - WiMAX, 3G, 4G) – 2.4, 5, 60 GHZ – До 15 Гбит/с

## **Передающая среда(физический уровень)**

Физический уровень описывает физические свойства (например, электромеханические характеристики) среды и сигналов, переносящих информацию. Это физические характеристики кабелей и разъемов, уровни напряжений и электрического сопротивления и.т.д., в том числе, например, спецификация кабеля «неэкранированная витая пара»

- Коаксиальный кабель(устарел) (До 500 м)
- Витая пара (от 10 Мбит/с до 100 Гбит/с)
- Оптика (~500 м(Multi-mode fiber), ~80км (Single mode))
- Wireless (До 15 Гбит/с)

## **Канальный уровень Ethernet**

Канальный уровень обеспечивает перенос данных по физической среде, обмен происходит кадрами. Уровень работает с применяемым в Ethernet физическими адресами, которые «вшиты» в сетевые адаптеры их производителями. Пример канального уровня: проверка доступности канала связи, если он общий для нескольких абонентов(Wireless).

- Управление доступом к среде (физическому уровню) передачи,
- Физическая адресация узлов (MAC)
- Обеспечение сервиса для протоколов более высокого уровня (Service Access Point)
- Упорядочивание кадров (фреймов), буферизация
- Учет топологии сети
- Управление потоком данных
- Определение способа взаимодействия источника и приемника (связь с установлением соединения или нет)

### Ethernet

| 7 байт                   | 1 байт            | 6 байт           | 6 байт          | 2 байта | 46 – 1500 байт | 4 байта           |
|--------------------------|-------------------|------------------|-----------------|---------|----------------|-------------------|
| Преамбула<br>(101010...) | SFD<br>(10101011) | Адрес получателя | Адрес источника | Тип     | данные         | Контрольная сумма |

**Преамбула** (Preamble) не несет полезной информации. Генерируемый при её передаче в физической среде сигнал извещает принимающие устройства о необходимости быть готовым к приему.

**SFD** (Start of Frame Delimiter) разделитель начала кадра тоже не несет полезной информации (в первых версиях разделитель считался частью преамбулы). Он позволяет приемнику точно определить момент начала передачи полезных данных.

Эти два элемента кадра явным образом предназначены для обеспечения правильной работы нижнего – физического уровня.

**Адрес получателя** DA (Destination Address) содержит уникальный MAC – адрес устройства, которому адресован данный кадр или специальный адрес для широковещательной рассылки пакетов.

### **MAC-адреса**

Этот адрес всегда состоит из 6 байт или 48 бит. Для его записи обычно используется шестнадцатеричная форма XX:XX:XX:XX:XX:XX.

Значения первых двух бит в первом байте - признак уникального адреса, остальные байты задают адрес конкретного сетевого адаптера. Уникальность адресации адаптеров обеспечивается специальным соглашением, по которому каждому производителю аппаратуры выделяется свое значение (одно или несколько) кода OUI (Organizationally Unique Identifier – уникальный идентификатор организации) — 22 бита из байтов 1-3. Байты 4-6 заполняются изготовителем — на нем лежит ответственность за их уникальность (эта информация может рассматриваться как серийный номер платы).

### **Сетевой уровень IP**

В отличии от канального уровня, имеющего дело с физическими адресами, сетевой уровень работает с логическими адресами. Он предназначается для определения пути передачи данных. Отвечает за трансляцию логических адресов и имён в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию, отслеживание неполадок и заторов в сети. Одна из основных функций сетевого уровня – маршрутизация, объединение нескольких разнородных локальных сетей в одну сеть.

К протоколам сетевого уровня относится IP и ICMP (Internet Control Message Protocol).

**IP-адрес** – уникальный сетевой адрес узла в компьютерной сети, построенной на основе стека протоколов TCP/IP

На сетевом уровне сам термин *сеть* наделяют специфическим значением. В данном случае под сетью понимается совокупность компьютеров, соединенных между собой в соответствии с одной из стандартных типовых топологий и использующих для передачи данных один из протоколов канального уровня, определенный для этой топологии.

Внутри сети доставка данных обеспечивается соответствующим канальным уровнем, а вот доставкой данных между сетями занимается сетевой уровень, который и поддерживает возможность правильного выбора маршрута передачи сообщения даже в том случае, когда структура связей между составляющими сетями имеет характер, отличный от принятого в протоколах канального уровня.

Сети соединяются между собой специальными устройствами, называемыми маршрутизаторами. *Маршрутизатор* – это устройство, которое собирает информацию о топологии межсетевых соединений и на ее основании пересыпает

пакеты сетевого уровня в сеть назначения. Чтобы передать сообщение от отправителя, находящегося в одной сети, получателю, находящемуся в другой сети, нужно совершить некоторое количество *транзитных передач между сетями, шаговое* (от *hop* — прыжок), каждый раз выбирая подходящий маршрут. Таким образом, маршрут представляет собой последовательность маршрутизаторов, через которые проходит пакет.

## Адресация, передача и маршрутизация пакетов

**Маршрутизация пакетов** проводится с помощью IP адресов. Маршрутизация осуществляется в посылающих TCP/IP конечных устройствах и в маршрутизаторах. В обоих случаях на интернет-уровне конечные устройства и в маршрутизаторы должны принимать решение куда отсылать пакет. Для принятия такого решения интернет-уровень получает информацию из специальных таблиц маршрутизации. Записи в таблице создаются автоматически при инициализации этого TCP/IP, хотя это может быть сделано и вручную. При доставке IP пакетов возможны два варианта доставки:

1. **Прямая доставка** (Direct delivery), в случае которой одно IP конечное устройство посыпает пакеты другому устройству, принадлежащему одному и тому же сектору, используя MAC адрес принимающего устройства.
2. **Косвенная доставка** (Indirect delivery) производится через промежуточные устройства или маршрутизаторы к цели, которая не относится к тому же сектору локальной сети. В этом случае посыпающий компьютер адресует пакет данных, используя MAC адрес маршрутизатора.

Более подробно ниже

**Маршрутизация** - процесс выбора пути для передачи пакета в сети. Под путем понимается последовательность маршрутизаторов, через которые проходит пакет по пути к узлу-назначению. IP-маршрутизатор – это специальное устройство, предназначенное для объединения сетей и обеспечивающее определение пути прохождения пакетов в составной сети.

Передача и адресация(надо дополнить)

### Прямая маршрутизация

Обычно мы ассоциируем маршрутизаторы с устройствами, которые действительно выполняют маршрутизацию, однако любое поддерживающее протокол IP устройство способно выполнять эту функцию . На рисунке узел 10 присоединен непосредственно к сети 10, и способен маршрутизировать пакеты к любому другому узлу в сети 10.

Узлу 10.1.1.1 необходимо передать пакет узлу 10.2.2.2. Первое, что он делает - определяет , находится ли IP-адрес получателя в одной с ним сети. Для этого сравнивает свой номер сети 10 с номером сети получателя 10. Делает вывод, что узел-получатель находится в одном с ним сегменте сети.

С помощью протокола ARP(Address Resolution Protocol) определяет MAC-адрес узла-получателя и посыпает пакет по этому адресу.

#### Косвенная маршрутизация

В следующем примере предполагается, что узел 10.1.1.1 имеет пакет, который нужно отправить узлу **172.16.0.1**.

1. Изучение адреса показывает, что узел назначения находится не на одной с передающим узлом сети. Узел 10.1.1.1 сконфигурирован так, что любые пакеты, требующие косвенной маршрутизации, передаются его шлюзу по умолчанию - маршрутизатору 1.
2. Чтобы доставить пакет маршрутизатору 1, узлу 10.1.1.1 необходим MAC-адрес маршрутизатора 10.3.3.3. Если MAC-адрес узлу 10.1.1.1 неизвестен, он отправляет ARP-запрос, чтобы его получить. Затем пакет, предназначенный для 172.16.0.1 отправляется маршрутизатору 1.
3. Маршрутизатор 1 осознает, что он подсоединен к сети 172.16. и полагает, что узел 172.16.0.1 должен быть частью этой сети. Маршрутизатор 1 реализует свою собственную процедуру прямой маршрутизации и посыпает ARP-запрос, ища узел назначения.

При отправке пакета от узла 10.1.1.1 до узла 192.168.1.1 узел-отправитель сравнивает номер своей сети с номером сети назначения и выясняет, что получатель находится в другой сети. Пакет будет отправлен шлюзу по умолчанию, в данном случае Маршрутизатору 1. Пусть в таблице маршрутизатора нет записи о сети 192.168.1, но шлюзом по умолчанию является маршрутизатор 2, тогда он перешлет пакет ему, а тот доставит пакет до получателя.

Если же узел 10.1.1.1 попытается отправить пакет узлу 192.164.1.1, то этот пакет будет перенаправлен маршрутизатору 1. Но так как он ничего не знает про сеть 192.164.1, то будет выполнено одно из самых главных правил маршрутизации: **если пакет получен, и таблица маршрутизации не содержит информации о его сети назначения, пакет следует отбросить. (И отправить ICMP сигнал об ошибке отправителю)**

В этой таблице в столбце

- Адрес сети назначения – указываются адреса всех сетей, которым данный маршрут может передавать пакеты
- Следующий маршрут в пути – IP-адрес удаленного маршрутизатора, которому необходимо послать пакет для доставки его по назначению
- Номер выходного порта – по которому следует отправить пакет

Расстояние до сети назначения – это любая метрика, используемая в соответствии с заданным в сетевом пакете классом сервиса. Это может быть количество хопов(переходов), время прохождения пакета по линиям связи, надежность линий связи или другая величина, отражающая качество данного маршрута по отношению к конкретному классу сервиса. Если маршрутизатор поддерживает несколько классов

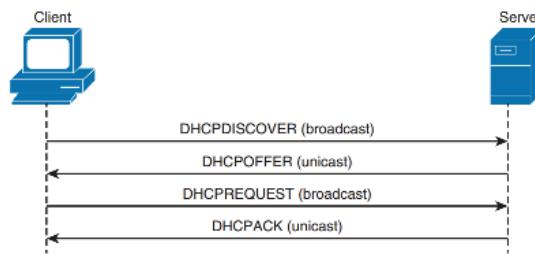
сервиса пакетов, то таблица маршрутов составляется и применяется отдельно для каждого вида сервиса (критерия выбора маршрута)

**Адресация пакетов** Каждый абонент (узел) локальной сети должен иметь свой уникальный адрес (идентификатор или MAC-адрес), для того чтобы ему можно было адресовать пакеты. Существуют две основные системы присвоения адресов абонентам сети (точнее, сетевым адаптерам этих абонентов). 1. Первая система сводится к тому, что при установке сети каждому абоненту пользователь присваивает индивидуальный адрес по порядку, к примеру, от 0 до 30 или от 0 до 254. Присваивание адресов производится программно или с помощью переключателей на плате адаптера. При этом требуемое количество разрядов адреса определяется из неравенства:  $2n > N_{\max}$  где  $n$  – количество разрядов адреса, а  $N_{\max}$  – максимально возможное количество абонентов в сети. Например, восемь разрядов адреса достаточно для сети из 255 абонентов. Один адрес (обычно 1111....11) отводится для широковещательной передачи, то есть он используется для пакетов, адресованных всем абонентам одновременно. Именно такой подход применен в известной сети Arcnet. Достоинства данного подхода – малый объем служебной информации в пакете, а также простота аппаратуры адаптера, распознающей адрес пакета. Недостаток – трудоемкость задания адресов и возможность ошибки (например, двум абонентам сети может быть присвоен один и тот же адрес). Контроль уникальности сетевых адресов всех абонентов возлагается на администратора сети. 2. Второй подход к адресации был разработан международной организацией IEEE, занимающейся стандартизацией сетей. Именно он используется в большинстве сетей и рекомендован для новых разработок. Идея этого подхода состоит в том, чтобы присваивать уникальный сетевой адрес каждому адаптеру сети еще на этапе его изготовления. Если количество возможных адресов будет достаточно большим, то можно быть уверенным, что в любой сети по всему миру никогда не будет абонентов с одинаковыми адресами. Поэтому был выбран 48-битный формат адреса, что соответствует примерно 280 триллионам различных адресов. Понятно, что столько сетевых адаптеров никогда не будет выпущено. \* Младшие 24 разряда кода адреса называются OUA (Organizationally Unique Address) – организационно уникальный адрес. Именно их присваивает каждый из зарегистрированных производителей сетевых адаптеров. Всего возможно свыше 16 миллионов комбинаций, то есть каждый изготовитель может выпустить 16 миллионов сетевых адаптеров. \* Следующие 22 разряда кода называются OUI (Organizationally Unique Identifier) – организационно уникальный идентификатор. IEEE присваивает один или несколько OUI каждому производителю сетевых адаптеров. Это позволяет исключить совпадения адресов адаптеров от разных производителей. Всего возможно свыше 4 миллионов разных OUI, это означает, что теоретически может быть зарегистрировано 4 миллиона производителей. Вместе OUA и OUI называются UAA (Universally Administered Address) – универсально управляемый адрес или IEEE-адрес. \* Два старших разряда адреса управляющие, они определяют тип адреса, способ интерпретации остальных 46 разрядов. Старший бит I/G (Individual/Group) указывает на тип адреса. Если он установлен в 0, то индивидуальный, если в 1, то групповой

(многопунктовый или функциональный). Пакеты с групповым адресом получат все имеющие этот групповой адрес сетевые адAPTERы. Причем групповой адрес определяется 46 младшими разрядами. Второй управляющий бит U/L (Universal/Local) называется флагком универсального/местного управления и определяет, как был присвоен адрес данному сетевому адAPTERу. Обычно он установлен в 0. Установка бита U/L в 1 означает, что адрес задан не производителем сетевого адAPTERа, а организацией, использующей данную сеть. Это случается довольно редко.

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

TCP/IP сеть позволяет пользовательским ЭВМ при подключении к сети в автоматическом режиме получить IP-адрес, маску, шлюз и стандартный DNS-сервер сети посредством DHCP сервера. Согласно протоколу DHCP, клиент отправляет широковещательный(его получат все узлы сети) запрос DHCPDISCOVER, чтобы найти DHCP сервер. Сервер, в ответ на данный запрос, отвечает предложением(DHCPOFFER) выдать адрес для клиента. Если клиент хочет воспользоваться этим DHCP сервером, то в ответ на DHCPOFFER он посыпает DHCPREQUEST, который тоже является широковещательным, чтобы уведомить все остальные DHCP сервера, что их услуги не требуются. После получения DHCPREQUEST, сервер резервирует за клиентом IP адрес и подтверждает право его использования пакетом DHCPACK.



Для удобства пользования сетью модель

TCP/IP предоставляет возможность создавать текстовые алиасы, называющиеся доменными именами, для хостов посредством сервиса DNS(Domain name system). Клиент, когда ему требуется получить IP-адрес, соответствующий имеющемуся доменному имени, отправляет запрос к DNS-серверу, и если у сервера есть информация о соответствии, то сервер отвечает на запрос и дает клиенту IP-адрес хоста. Если у сервера нет информации о соответствии, то он обращается к вышестоящему DNS-серверу и, кроме того, чтобы отправить полученную информацию клиенту, кэширует её и в следующий раз даст клиенту информацию не обращаясь к вышестоящим серверам. Существуют DNS-сервера самого высокого уровня, которые называются корневыми.

Транспортный уровень модели OSI (равно как и стека протоколов TCP/IP) обеспечивает следующие возможности:

- Отслеживание индивидуальных сеансов общения между приложениями на передающем и принимающем устройствах.  
Эта функция относится только к протоколу TCP. Перед началом передачи полезных данных узел, инициирующий соединение открывает сессию с узлом получателем, чтобы убедиться, что получатель существует и готов принимать данные. Далее все полезные данные передаются в рамках установленной сессии и после завершения такой передачи сессия закрывается.
- Сегментация данных (разбиение больших порций данных на сегменты для индивидуальной отсылки по сети, сборка этих сегментов после получения)
- Идентификация приложений, передающих и принимающих данные.  
Для идентификации приложений используются номера портов: каждое приложение, желающее работать с сетью сообщает операционной системе о своих планах и регистрирует за собой какой-то номер порта. Впоследствии, когда на компьютер придут данные, транспортный уровень заглянет в поле «номер порта получателя» и передаст эти данные соответствующему приложению.

Прикладной уровень (application layer) — это в действительности просто набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам, таким как файлы, принтеры или гипертекстовые веб-страницы, а также организуют свою совместную работу, например, по протоколу электронной почты. Единица данных, которой оперирует прикладной уровень, обычно называется сообщением.

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

### **Интерфейсы ввода-вывода**

Интерфейсы определяют конкретные детали обмена(Частота, набор каналов передачи, способ кодирования, команды, представление данных, набор данных и последовательность...)

Требуется аппаратная и программная реализация. Нуждаются в точной спецификации и стандартизации(стороны обмена должны однозначно интерпретировать детали обмена)

### **Уровни стандартизации**

Логическое подключение (все алгоритмы неким образом понимаются системой)

Физические параметры сигналов (напряжение, ток)

Конструктивные особенности (физическая реализация разъёмов, портов и т.п.)

Сопряжение с системной шиной происходит через контроллер ВУ. Два уровня сопряжения: Процессор <-> Контроллеры; Контроллеры <-> ВУ

Управляющие сигналы позволяют организовать 2 режима обмена информацией с контроллерами ВУ:

1. Программно-управляемый (асинхронный и по прерыванию)
2. Прямой доступ в память

### **Сопряжение с системной шиной**

При использовании для обмена с ВУ команд ввода-вывода адрес (номер) ВУ передается пошине адреса. По этой жешине передаются и адреса ячеек памяти. Информация нашине адреса имеет смысл адреса ВУ только при наличии специальных управляющих сигналов (например, «Ввод из ВУ», «Вывод в ВУ», которые инициируются соответствующими командами ввода-вывода).

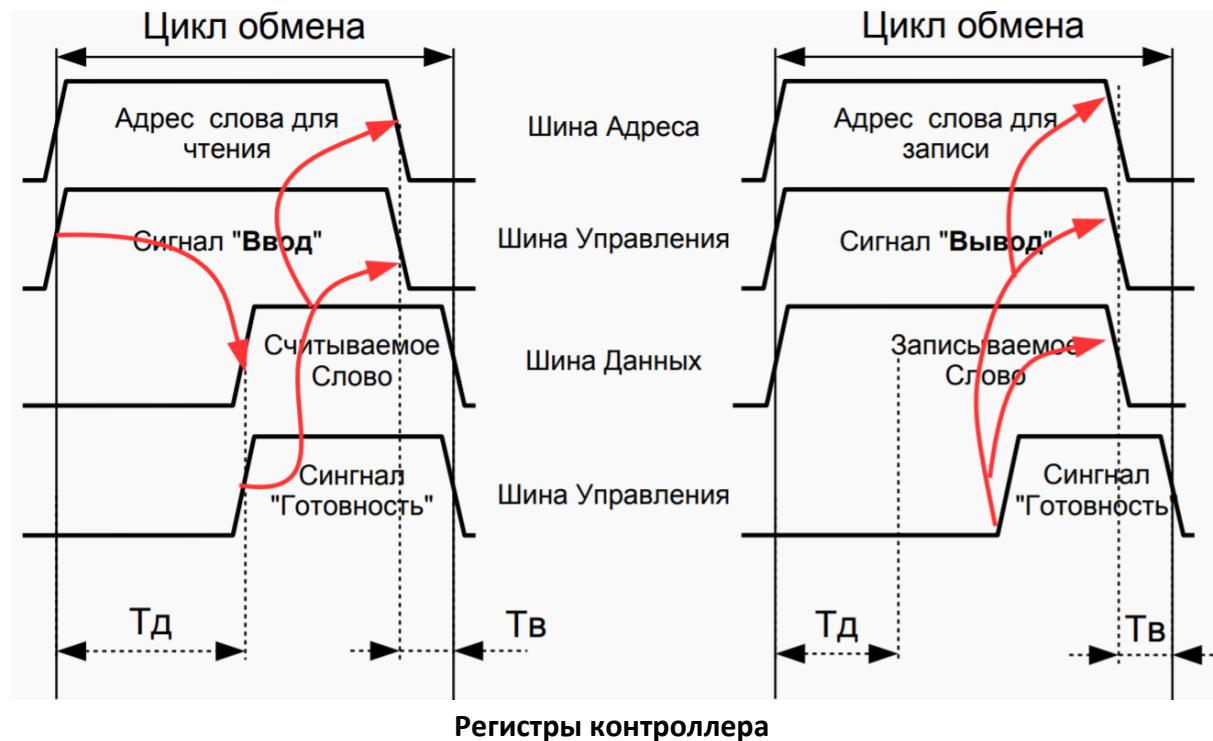
При реализации обмена с ВУ по аналогии с обращениями к памяти нет необходимости в спец. сигналах, указывающих, что нашине адреса находится адрес ВУ. Для этих адресов отведена часть адресного пространства, в контроллерах организована селекция адресов ВУ. Но остается необходимость передавать в ВУ приказ на ввод/вывод информации. Для этого есть линии управляющей шины «Чтение» и «Запись». Они обеспечивают обмен информацией микропроцессора с модулями памяти.

### **Циклы обмена**

Операция вывод: микропроцессор выставляет налиниях адресной шины адрес (номер) ВУ, налиниях шины данных – значения разрядов выводимого слова данных и единичным сигналом по линии «Вывод в ВУ» указывает тип операции. Адресуемый

контроллер ВУ принимает данные, пересыпает их в ВУ и единичным сигналом по линии «Готовность ВУ» сообщает процессору, что данные приняты в ВУ и можно снять информацию с шин адреса и данных, а также сигнал «Вывод в ВУ».

Операция ввод: микропроцессор выставляет на линиях адресной шины адрес (номер) ВУ, и единичным сигналом на линии «Ввод из ВУ» указывает тип операции. По сигналу «Ввод из ВУ» контроллер адресуемого ВУ считывает слово данных из ВУ, выставляет на линиях шины данных значения разрядов считанного слова и единичным сигналом на линии «Готовность ВУ» сообщает об этом процессору. Приняв данные, процессор снимает сигналы с шины адреса и линии «Ввод из ВУ».



Рассмотрим типичные структуры контроллеров ВУ:

В основе контроллера несколько регистров (для временного хранения передаваемой информации). Каждый регистр имеет свой адрес (такие регистры называют портами ввода-вывода).

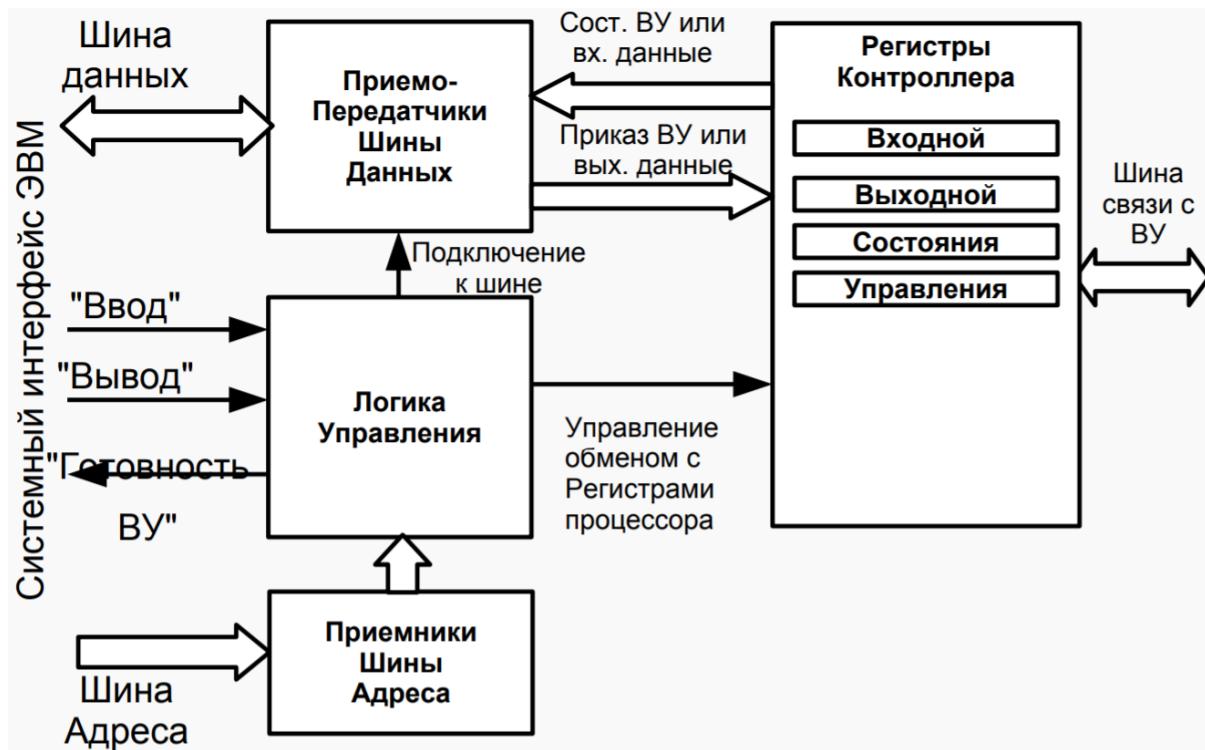
*Регистр входных данных* работает в режиме чтения, *регистр выходных данных* в режиме записи.

*Регистр состояния* в режиме чтения и содержит информацию о состоянии ВУ (включено/выключено, готово/не готово к обмену данными).

*Регистр управления* в режиме записи, служит для приема из микроЭВМ приказов для ВУ.

Логика управления выполняет селекцию адресов регистров контроллера, прием, обработку и формирование управляющего сигнала системного интерфейса, обеспечивая обмен информацией между регистрами контроллера и шиной данных системного интерфейса.

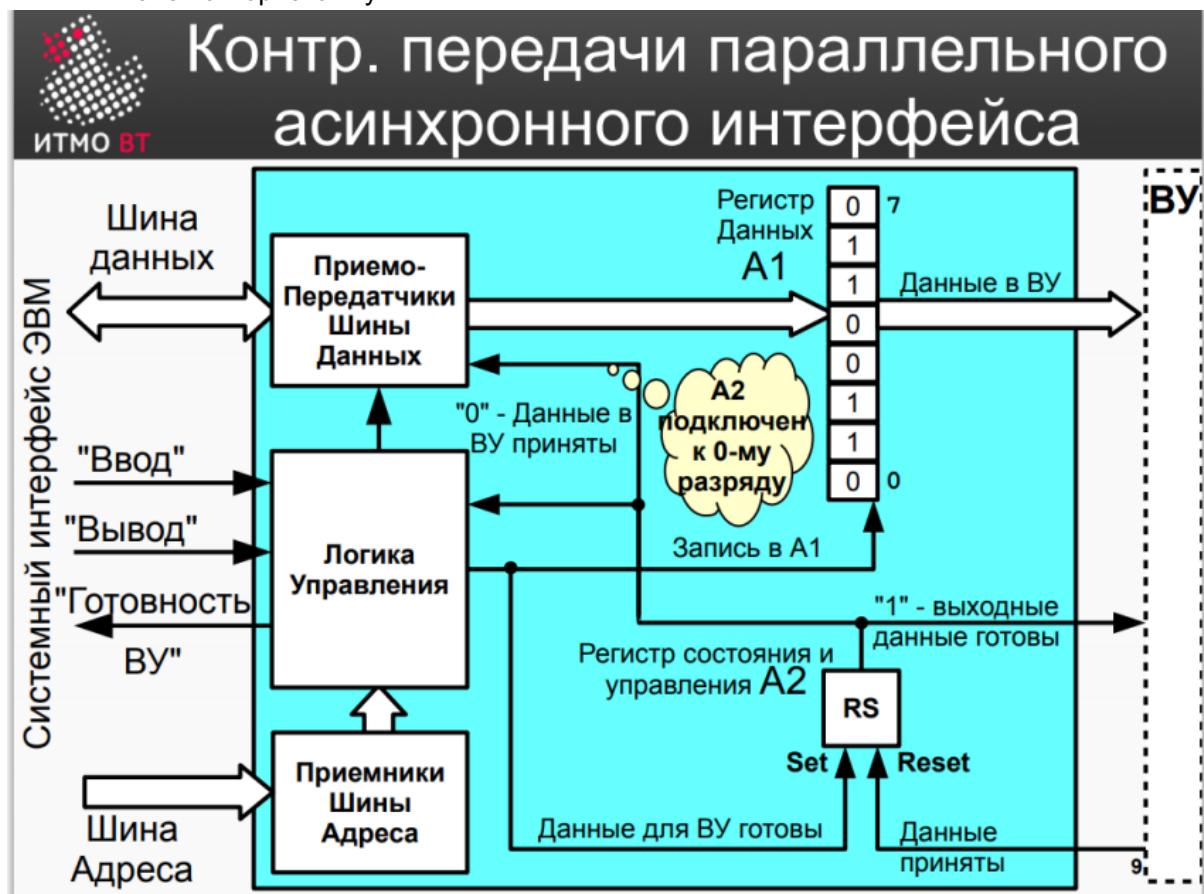
Приемопередатчики шин адреса и данных – для физического подключения электронных схем контроллера к шинам системного интерфейса.



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

Параллельная передача данных в ВУ под управлением программы асинхронного обмена:

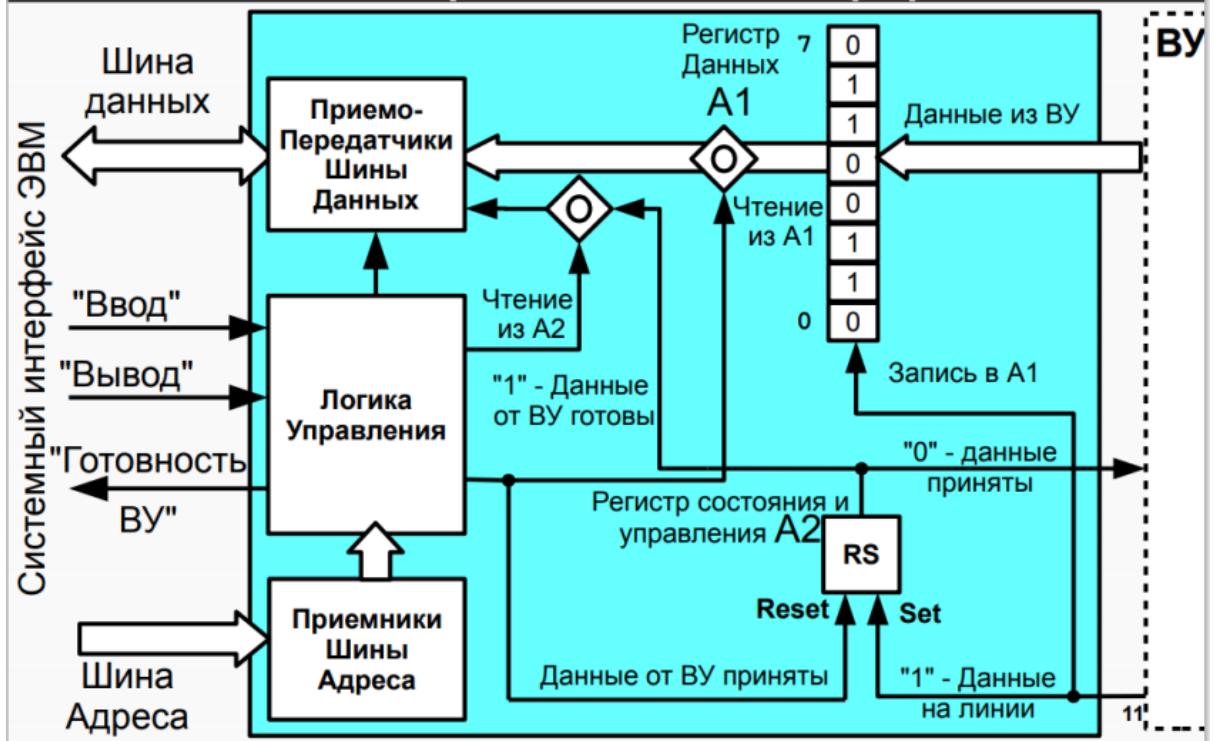
1. Процессор микроЭВМ проверяет готовность ВУ к приему данных
2. Если ВУ готово к приему данных (логический 0 в регистре A2), то данные передаются из шины данных системного интерфейса в регистр данных A1 контроллера и далее в ВУ. Иначе повторяется пункт 1.



В шине связи с ВУ используются 2 управляющих сигнала. Для формирования управляющего сигнала «Выходные данные готовы» и приема из ВУ управ. сигнала «Данные принятые» в контроллере используется одноразрядный адресуемый регистр состояния и управления A2. Одновременно с записью очередного байта данных из шины данных сист. интерфейса в адресуемый регистр данных контроллера A1 в регистр состояния и управления записывается логическая единица (формируется управляющий сигнал «Выходные данные готовы»). ВУ, приняв байт данных, управ. сигналом «Данные принятые» обнуляет регистр состояния. При этом формируется:

- Управ. сигнал сист. интерфейса «Готовность ВУ»
- Признак готовности ВУ к обмену, передаваемый в процессор по одной из линий шины данных В логике управления – селекция адресов регистров контроллера, прием и формирование управ. сигналов, формирование сигнала «Готовность ВУ»
- Для сопряжения регистров контроллера с шинами адреса и данных сист. интерфейса используются приемники шины адреса и приемопередатчики шины данных

# Контр. приема параллельного асинхронного интерфейса

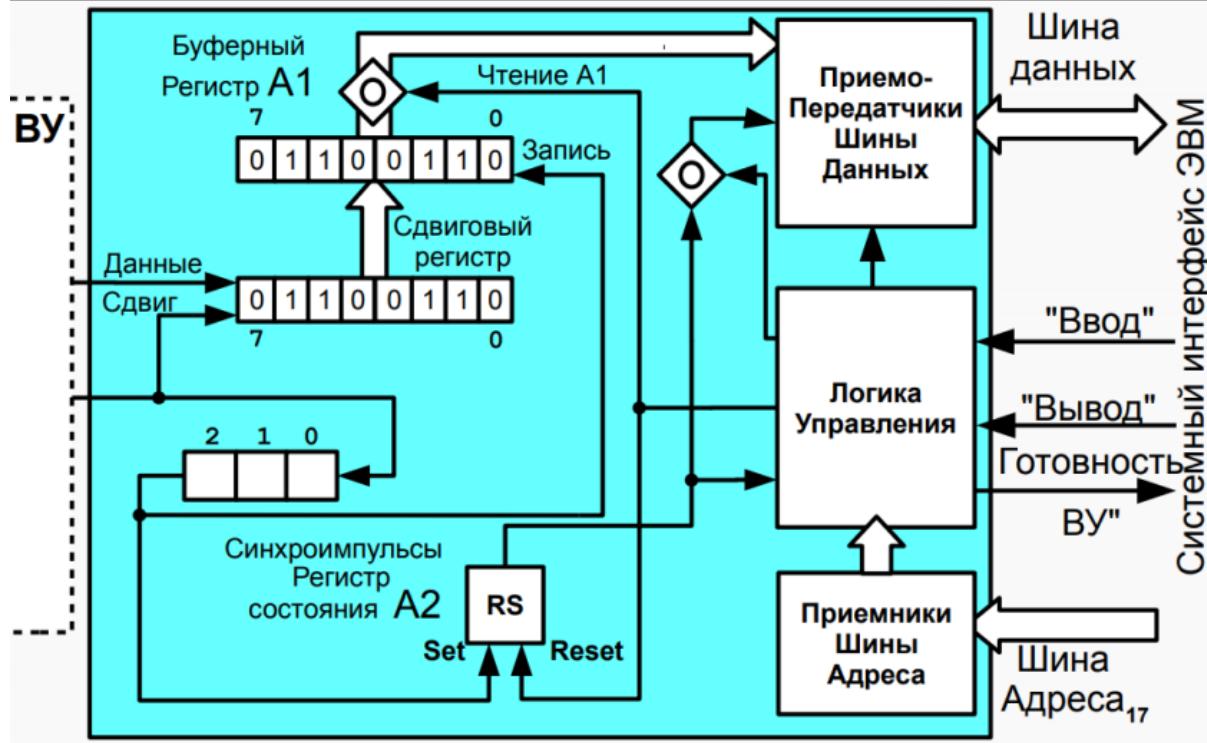


36. Синхронные последовательные интерфейсы. Контроллеры последовательной передачи и приема.



- 8-ми разрядный буферный регистр контроллера A1 - для временного хранения байта данных до его загрузки в сдвиговый регистр
- Запись байта данных в буферный регистр происходит при наличии 1 в регистре состояния A2
- 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 A1 обнуляется регистр A2
- В сдвиговом регистре происходит преобразование данных из параллельного формата в последовательный и передача их в линию связи
- По очередному тактовому импульсу содержимое сдвигового регистра сдвигается на 1 разряд вправо и в линию связи «Данные» выдается значение очередного разряда
- Одновременно со сдвигом по линии «Синхронизация» передается тактовый импульс. Количество переданных в линию тактовых сигналов (переданных бит) подсчитывается счетчиком тактовых импульсов. Как только его содержимое равно 7(передано 8 бит информации) формируется управляющий сигнал «Загрузка» и происходит запись в сдвиговый регистр очередного байта из буферного. Устанавливается в 1 регистр состояния
- Следующим тактовым импульсом счетчик будет сброшен в 0 и начнется очередной цикл выдачи 8 бит из сдвигового регистра в линию связи.

# Контр. приема синхронного последовательного интерфейса



- Буферный регистр контроллера A1 - для временного хранения байта , поступившего из сдвигового регистра.
- Чтение байта данных из буферного регистра происходит при наличии 1 в регистре состояния A2
- Данные, поступающие из линии связи в последовательном коде преобразуются в параллельный с помощью сдвигового регистра
- Линия «Данные» подключается в контроллере к последовательному входу сдвигового регистра, а линия «Синхронизация» - на управ. вход «Сдвиг» и на вход счетчика тактовых импульсов.
- По тактовому импульсу по линии «Синхронизация» производится сдвиг содержимого сдвигового регистра на 1 разряд влево и запись очередного бита данных из линии «Данные» в младший разряд этого регистра. Одновременно увеличивается на 1 счетчик тактовых импульсов.
- Как только он становится равным 7, формируется управ. сигнал «Запись» и происходит запись в буферный регистр байта из сдвигового. Устанавливается в 1 регистр состояния.
- При передаче байта данных из буферного регистра в шину данных регистр состояния обнуляется (т.е. в сдвиговый регистр принимается очередной байт информации).

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

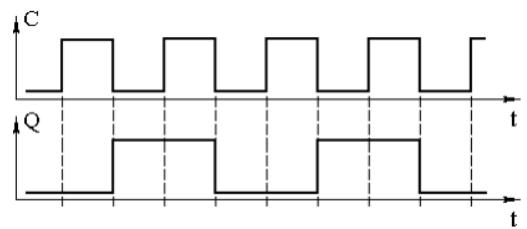
При реализации асинхронного обмена интервал между командами передачи данных задается самим внешним устройством. Контроллеры этих устройств снабжаются регистром состояния, который информирует ЭВМ о готовности устройства к обмену информацией. Обмен происходит по готовности ВУ.

Преимущества:

- не нужно знать время выполнения операции на ВУ
- ВУ всегда успеет выполнить обработку данных перед началом следующей операции обмена

Недостаток: ЭВМ не выполняет никаких полезных действий во время ожидания момента готовности ВУ

При асинхронной передаче данных, со временем может происходить рассинхронизация генераторов тактов передатчика и приемника, 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 времени передачи данных, что увеличивает точность передачи. В этом примере частоту поделили на 2. На примере понять, как работает деление частоты, можно в некст вопросе.



Очень часто необходимо использовать триггер для деления частоты входной последовательности импульсов на два, т. е. производить переключение триггера в новое состояние каждым входным импульсом (фронтом или спадом). Такой триггер называют счетным, или Т-триггером.

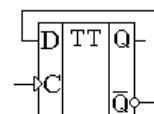
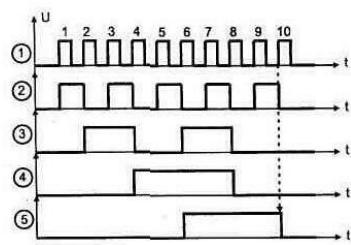
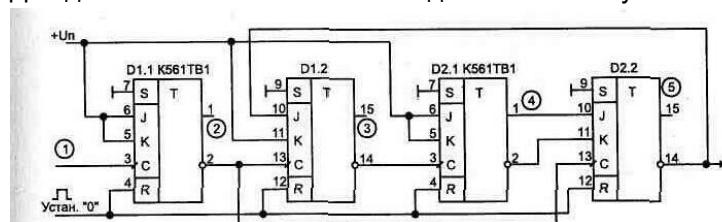


Схема Т триггера, построенная на основе D триггера

Для деления частоты на степень двойки используется каскад из Т-триггеров:



Формат кадра – количество бит стартовых бит, количество бит информации, количество стоповых бит. В кадре может содержаться дополнительная информация, такая, как длина кадра и тд.

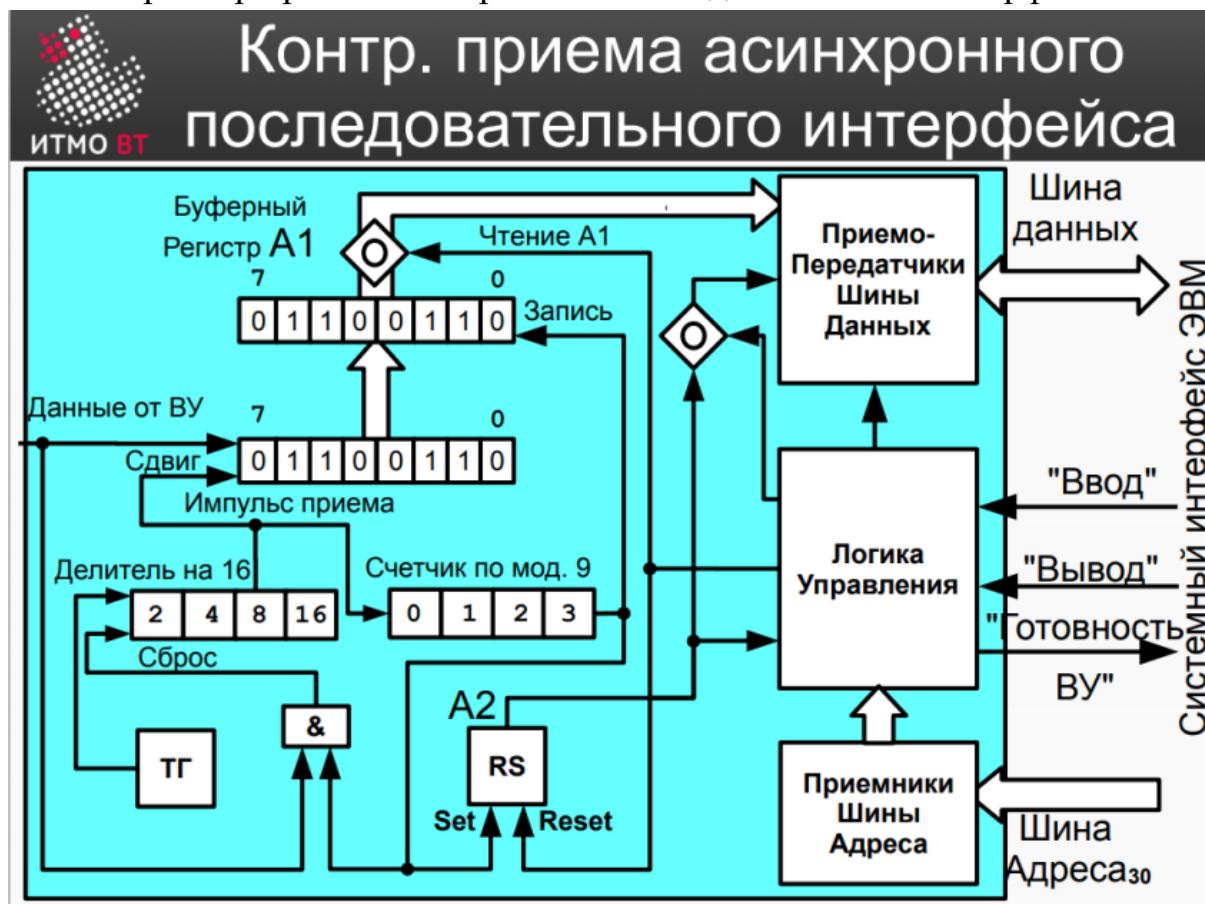
38. Контроллер передачи асинхронного последовательного интерфейса.



- После передачи предыдущих байтов данных в Регистр Состояния A2 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.
- Процессор, выполняя команду «Вывод», выставляет передаваемый байт на шине данных и формирует управляющий сигнал системного интерфейса «Вывод».
- По сигналу «Вывод» в контроллере происходит запись передаваемого байта в буферный регистр A1, сброс регистра состояния A2 и формирование на вентиле И сигнала «Загрузка».
- 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
- 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)
- 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.
- 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 A2. Единичный сигнал с выхода регистра A2 запретит формирование импульсов сдвига, и информирует о готовности к приему нового байта данных.

- После завершения передачи очередного кадра (стартового бита, информационного бита и двух стоповых битов), на линии передачи данных поддерживается значение второго стопового бита – единицы

39. Контроллер приема асинхронного последовательного интерфейса.



На линии данных от ВУ находится единица, что запрещает работу делителя частоты генератора тактовых импульсов.

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.

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.

Последующие сдвиги происходят после прохождения 16-ти тактовых импульсов.

При приеме в сдвиговый регистр 9-го бита кадра (8-го инф. Бита), из него выдвинется стартовый бит, и, следовательно в сдвиговом регистре будет размещен информационный байт. В этот момент счетчик импульсов сдвига придет в нулевое состояние и на его выходе будет выработан единичный сигнал, это приведет к тому, что:

- содержимое сдвигового регистра будет переписано в буферный регистр
- в триггер A2 запишется 1 и он будет информировать процессор об окончании приема очередного байта
- вентиль И подготовится к выработке сигнала «Сброс». Он сформируется после прихода первого стопового бита
- Получив сигнал готовности (1 в триггере A2), 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 триггера A2

## 40. Организация прямого доступа к памяти. Контроллер ПДП.

**Прямой доступ к памяти** ([англ.](#) *direct memory access, DMA*) — режим обмена данными между устройствами [компьютера](#) или же между устройством и [основной памятью](#), в котором [центральный процессор](#) (ЦП) не участвует. Так как данные не пересыпаются в ЦП и обратно, скорость передачи увеличивается.

Этот тип обмена реализуется полностью аппаратно и управляет контроллером ПДП.

### Организация ПДП

#### Особенности ПДП

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

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

Эта проблема решается двумя способами:

1. Захват цикла
  - a. Простой захват цикла.

Передача происходит в те машинные циклы, в которых процессор не обменивается данными с памятью. Пометка таких циклов выполняется либо с помощью спец. указывающего цикла, либо такие циклы выбираются с помощью спец. селектирующих схем, что усложняет конструкцию процессоров.

При таком способе организации обмена ПДП не снижает производительности микроЭВМ, но обмен возможно только в случайные моменты времени одиночными байтами или словами.
  - b. Захват цикла с принудительным отключением процессора от шин системного интерфейса.

Для его реализации такого режима ПДП системный интерфейс (СИ) дополняется двумя линиями для передачи управляющих сигналов «Требование ПДП» (ТПДП) и «Предоставление ПДП» (ППДП).

1. ТПДП формируется контроллером ПДП.

2. Получив сигнал ТПДП, процессор приостанавливает выполнение очередной команды, не дожидаясь её завершения, выдает в системный интерфейс ППДП и отключается от шин СИ
  3. Контроллер ПДП получает управления над шинами СИ и осуществляет обмен одним байтом или словом данных с памятью микроЭВМ.
  4. Контроллер ПДП возвращает управление СИ процессору.
2. Блокировка процессора
- Отличается от «Захвата цикла» тем, что управление СИ передается контроллеру ПДП не на время обмена одним байтом, а на время обмена блоком данных.
- Контроллер ПДП ввода данных из ВУ в режиме «Захват цикла»**
1. Процессор загружает в СК контроллера количество принимаемых байтов, а в РА контроллера начальный адрес области памяти для вводимых данных.
  2. Байты данных из ВУ поступают в РД контроллера, при этом каждый байт сопровождается управляющим сигналом из ВУ «Ввод данных», который обеспечивает запись байта в РД контроллера. По этому же сигналу при ненулевом состоянии СК контроллер формирует сигнал ТПДП.
  3. По ответному сигналу процессора ППДП контроллер выставляет на ША и ЩД содержимое своих РА и РД.
  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

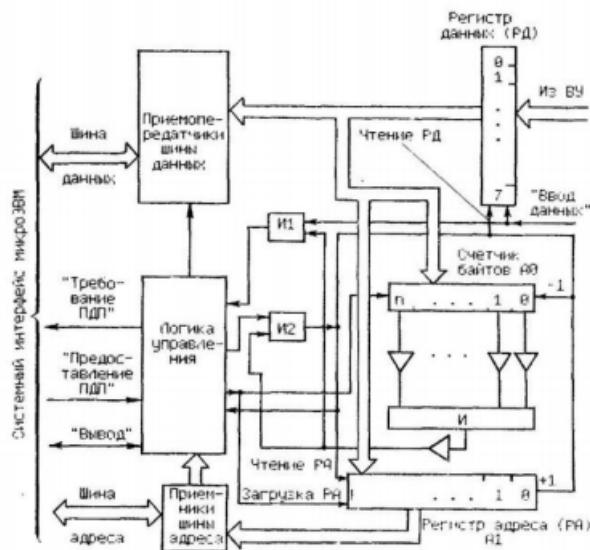


Рис. 8.16. Контроллер ПДП для ввода данных из ВУ в режиме «Захват цикла»