Мадока ВТ Вики

ОПД:Экзамен Сем2

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

Две формы представления информации

Вся информация может быть представлена в двух основных формах: аналоговой и цифровой (дискретной).

1. **Аналоговая информация:** Представлена непрерывной величиной, которая может принимать любое значение в заданном диапазоне.
   - **Примеры:** Напряжение в электрической цепи, высота горы, температура воздуха.
   - **Аналоговые ЭВМ:** Выполняли вычисления, оперируя непрерывными физическими величинами (например, напряжением). Они были быстрыми для решения определённых классов задач (например, дифференциальных уравнений), но страдали от низкой точности и больших габаритов. Пример из лекции — логарифмическая линейка.
2. **Дискретная (цифровая) информация:** Представлена в виде набора отдельных, конечных значений. Цифровые компьютеры используют именно эту форму.
   - **Принцип:** Весь диапазон возможных значений аналогового сигнала разбивается на конечное число уровней. В современных цифровых ЭВМ используется два уровня, кодируемые как **0** и **1**. Реальный сигнал (например, напряжение) считается “1”, если он попадает в один диапазон напряжений, и “0”, если в другой.
   - **Преимущества:** Высокая точность, помехоустойчивость, универсальность.

Системы счисления

Система счисления (СС) — это способ записи чисел с помощью набора символов (цифр). Ключевой характеристикой является **основание** системы — количество цифр, используемых для представления чисел.

- **Десятичная (основание 10):** Использует цифры 0-9. Привычная для человека.
- **Двоичная (основание 2):** Использует только цифры 0 и 1. Это фундаментальная система для всех цифровых компьютеров, так как легко представляется двумя состояниями (включено/выключено, есть напряжение/нет напряжения).
- **Восьмеричная (основание 8):** Использует цифры 0-7. Является компактной формой записи двоичных чисел, так как одна восьмеричная цифра представляет собой три двоичных разряда (триаду). Например, $111_2 = 7_8$.
- **Шестнадцатеричная (основание 16):** Использует цифры 0-9 и буквы A-F (где A=10, B=11, …, F=15). Еще более компактная форма записи двоичных чисел. Одна шестнадцатеричная цифра представляет четыре двоичных разряда (тетраду). Например,

$1111_2 = F_{16}$. Широко используется в программировании для представления адресов памяти, кодов команд, цветов и т.д.

- **Двоично-десятичная (BCD - Binary-Coded Decimal):** Каждая десятичная цифра кодируется своей собственной четырехбитной двоичной последовательностью. Например, число 91_{10} будет представлено как `1001 0001`. Используется в некоторых устройствах, например, калькуляторах, для упрощения вывода на десятичные дисплеи.

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

Представление чисел с фиксированной точкой

Это способ представления чисел, при котором положение двоичной точки (аналога десятичной запятой) жестко зафиксировано. Для целых чисел точка находится после младшего разряда. Для дробных — в любом другом месте.

Беззнаковые числа: Все разряды используются для представления величины числа. В БЭВМ (16 разрядов) диапазон от 0 до 65535 (2^{16}-1).

Знаковые числа: Старший разряд (в БЭВМ — 15-й) отводится под знак: **0** для положительных чисел, **1** для отрицательных.

Способы кодирования знаковых чисел

1. **Прямой код (Sign-Magnitude):** Простейший способ. Старший бит — знак, остальные — абсолютная величина числа.
   - **Недостаток:** Существует два представления нуля (+0 и -0), что усложняет арифметические операции.
2. **Дополнительный код (Two's Complement):** Основной способ представления знаковых чисел в современных компьютерах.
   - **Положительные числа:** Кодируются так же, как в прямом коде (старший бит 0).
   - **Отрицательные числа:** Для получения дополнительного кода отрицательного числа нужно:
     1. Взять прямой код его модуля (положительного значения).
     2. Инвертировать все биты (получить обратный код).
     3. Прибавить к результату единицу.
   - **Преимущества:**
     - Только одно представление нуля.
     - Операция вычитания (A - B) сводится к сложению с числом в дополнительном коде (A + (-B)), что значительно упрощает схему арифметико-логического устройства (АЛУ).

Битовые признаки результата (флаги)

После выполнения арифметической операции в регистре состояния (PS) устанавливаются флаги, характеризующие результат.

- **N (Negative) — Признак знака:** Копирует значение знакового (старшего) бита результата. Если N=1, результат отрицательный.
- **Z (Zero) — Признак нуля:** Устанавливается в 1, если результат операции равен нулю.
- **C (Carry) — Признак переноса:** Устанавливается в 1, если при операции с **беззнаковыми** числами произошел перенос из старшего разряда. Сигнализирует о выходе результата за пределы беззнакового диапазона.
- **V (oVerflow) — Признак переполнения:** Устанавливается в 1, если при операции со **знаковыми** числами результат вышел за пределы допустимого диапазона. Это происходит, когда:
  - Складываются два положительных числа, а результат отрицательный.
  - Складываются два отрицательных числа, а результат положительный.
  - Аппаратно это определяется как $V = C_{14} \oplus C_{15}$ (в БЭВМ, где C_{15} - перенос из знакового разряда, а C_{14} - в знаковый разряд).

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

Представление символов

Символы представляются в памяти компьютера в виде числовых кодов согласно **кодовой таблице** (кодировке).

- **ASCII (American Standard Code for Information Interchange):** Исторически первый и самый важный стандарт. Это 7-битная кодировка, содержащая 128 символов: латинские буквы (прописные и строчные), цифры, знаки препинания и управляющие символы (например, перевод строки). В 8-битных системах старший бит часто использовался для контроля четности или для расширений.
- **8-битные кодировки:** Для представления национальных алфавитов (например, кириллицы) были созданы расширения ASCII.
  - **КОИ-8 (Код Обмена Информацией, 8-битный):** Популярная кодировка для кириллицы в ранних UNIX-системах.
  - **Windows-1251:** Стандартная кодировка для кириллицы в Microsoft Windows.
  - **ISO 8859-5:** Международный стандарт для кириллицы.
  - **Проблема:** Множество несовместимых 8-битных кодировок приводило к проблемам с отображением текста (“кракозябрам”).
- **Unicode:** Глобальный стандарт, цель которого — присвоить уникальный код каждому символу из всех существующих письменных языков мира. Сам стандарт определяет только числовые коды (code points).
- **Кодировки Unicode (UTF - Unicode Transformation Format):** Определяют, как числовые коды Unicode представляются в виде последовательности байтов.
  - **UTF-8:** Самая популярная кодировка в интернете. Имеет переменную длину символа (от 1 до 4 байт). Полностью совместима с ASCII: символы с кодами 0-127 кодируются одним байтом, как в ASCII.
  - **UTF-16:** Кодировка с переменной длиной (2 или 4 байта). Является основной в операционных системах Windows (начиная с NT) и в языке Java.

Представление строковых данных

Строка — это последовательность символов. В памяти она может храниться несколькими способами:

1. **NUL-terminated String (строка, завершающаяся нулем):** Строка представляет собой массив символов, конец которого обозначается специальным символом с кодом 0 (NUL). Этот подход используется в языке C.
   - **Преимущество:** Простота.
   - **Недостаток:** Чтобы узнать длину строки, нужно пройти по ней до конца. Сам символ NUL не может быть частью строки.
2. **Length-prefixed String (строка с префиксом-длиной):** Перед данными строки хранится ее длина (например, в первом байте или слове). Такой подход использовался в языке Pascal.
   - **Преимущество:** Длина строки известна сразу.
   - **Недостаток:** Длина строки ограничена размером поля длины.
3. **Порядок байт (Endianness):** При работе с многобайтными кодировками (UTF-16) и строками, упакованными в машинные слова, важен порядок байт в памяти.
   - **Big-endian:** Старший байт хранится по младшему адресу (как мы привыкли читать числа).
   - **Little-endian:** Младший байт хранится по младшему адресу (используется в процессорах Intel x86).

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

- **Ячейка/Регистр:** Элемент для хранения одного бита или слова данных. Ячейки образуют память, регистры — сверхоперативную память внутри процессора.
- **Логические схемы (элементы):** Базовые “кирпичики” для выполнения логических операций: И (AND), ИЛИ (OR), НЕ (NOT), Исключающее ИЛИ (XOR) и их инверсии (И-НЕ, ИЛИ-НЕ). Из них строятся все более сложные узлы компьютера.
- **Триггер:** Электронная схема с двумя устойчивыми состояниями, способная хранить 1 бит информации. Является основой для построения регистров и статической памяти (SRAM).
- **Счетчик:** Схема, которая подсчитывает количество входных импульсов. Строится на основе триггеров.
- **Сумматор:** Схема для выполнения арифметического сложения двоичных чисел. Одноразрядный сумматор имеет три входа (два слагаемых бита и бит переноса с предыдущего разряда) и два выхода (бит суммы и бит переноса на следующий разряд).
- **Вентиль:** Управляемый элемент, который либо пропускает сигнал, либо блокирует его. Позволяет подключать выходы различных регистров к общей шине по очереди. Реализуется на основе логического элемента И.
- **Шина (Bus):** Группа проводников, по которым передаются данные, адреса и управляющие сигналы между различными блоками компьютера (процессор, память, контроллеры).
- **Тактовый генератор:** Схема, генерирующая последовательность электрических импульсов (тактов) с постоянной частотой. Эти импульсы синхронизируют работу всех узлов компьютера, задавая ритм выполнения операций.

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

Общая структура ЭВМ включает:

- **Центральный процессор (CPU):** “Мозг” компьютера.
- **Арифметико-логическое устройство (АЛУ):** Выполняет арифметические и логические операции.
- **Устройство управления (УУ):** Руководит работой всех блоков ЭВМ, формируя управляющие сигналы на основе команд программы.
- **Память (ОЗУ):** Хранит программы и данные.
- **Устройства ввода-вывода (УВВ):** Обеспечивают взаимодействие с пользователем и внешним миром.

Принцип работы простого процессора (на примере калькулятора): Работа процессора — это циклическое выполнение последовательности простых действий (микроопераций), синхронизированных тактовым генератором. Рассмотрим сложение `7 + 5`.

1. **Ввод первой цифры (7):**
   - Нажимается клавиша “7”.
   - Устройство управления (УУ) генерирует сигналы, которые заносят код цифры 7 в регистр X (регистр операнда).
2. **Ввод знака операции (+):**
   - Нажимается клавиша “+”.
   - УУ пересылает содержимое регистра X (число 7) в регистр Y (регистр-аккумулятор), а регистр X очищается. Код операции “+” запоминается.
3. **Ввод второй цифры (5):**
   - Нажимается клавиша “5”.
   - УУ заносит код цифры 5 в регистр X.
4. **Выполнение операции (=):**
   - Нажимается клавиша “=”.
   - УУ подает содержимое регистров X (5) и Y (7) на входы АЛУ.
   - УУ подает на АЛУ команду “сложить”.
   - АЛУ выполняет сложение, результат (12) записывается обратно в регистр X.
   - Содержимое регистра X (12) отображается на индикаторе.

Этот пример демонстрирует, как сложная задача разбивается на последовательность элементарных шагов (микроопераций), управляемых сигналами от УУ в такт с генератором.

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

Ядро ОС (Operating System Kernel)

Ядро — это центральная часть операционной системы, которая управляет ресурсами компьютера и предоставляет приложениям (пользовательским процессам) абстрактный и безопасный интерфейс для работы с этими ресурсами. В Unix-подобных системах ядро можно представить состоящим из нескольких ключевых подсистем:

1. **Интерфейс системных вызовов (System Call Interface):** Это граница между пользовательскими программами и ядром. Когда программе нужно выполнить привилегированную операцию (например, прочитать файл или отправить данные по сети), она делает системный вызов. Ядро перехватывает этот вызов, переключает процессор в режим ядра, выполняет требуемое действие и возвращает результат программе.
2. **Управление процессами и потоками:** Эта подсистема отвечает за создание, планирование и уничтожение процессов и потоков. Она распределяет процессорное время между ними, создавая иллюзию одновременной работы множества программ.
3. **Управление памятью:** Управляет оперативной памятью (ОЗУ). Выделяет память процессам, отслеживает ее использование и освобождает после завершения процесса. Важнейшей функцией является реализация **виртуальной памяти**, которая изолирует адресные пространства процессов друг от друга и позволяет использовать больше памяти, чем физически установлено (за счет области подкачки на диске).
4. **Виртуальная файловая система (VFS):** Предоставляет единый интерфейс для работы с различными типами файловых систем (на жестком диске, сетевых, виртуальных). Благодаря VFS, пользовательские программы работают с файлами одинаково, не задумываясь о том, где и как они физически хранятся.
5. **Сетевая подсистема:** Реализует стек протоколов (например, TCP/IP), позволяя компьютеру обмениваться данными с другими устройствами в сети.
6. **Драйверы устройств:** Это модули ядра, которые управляют конкретными аппаратными устройствами (видеокартой, жестким диском, сетевой картой). Они “переводят” высокоуровневые запросы от других подсистем ядра в низкоуровневые команды, понятные аппаратуре.

Файловая система

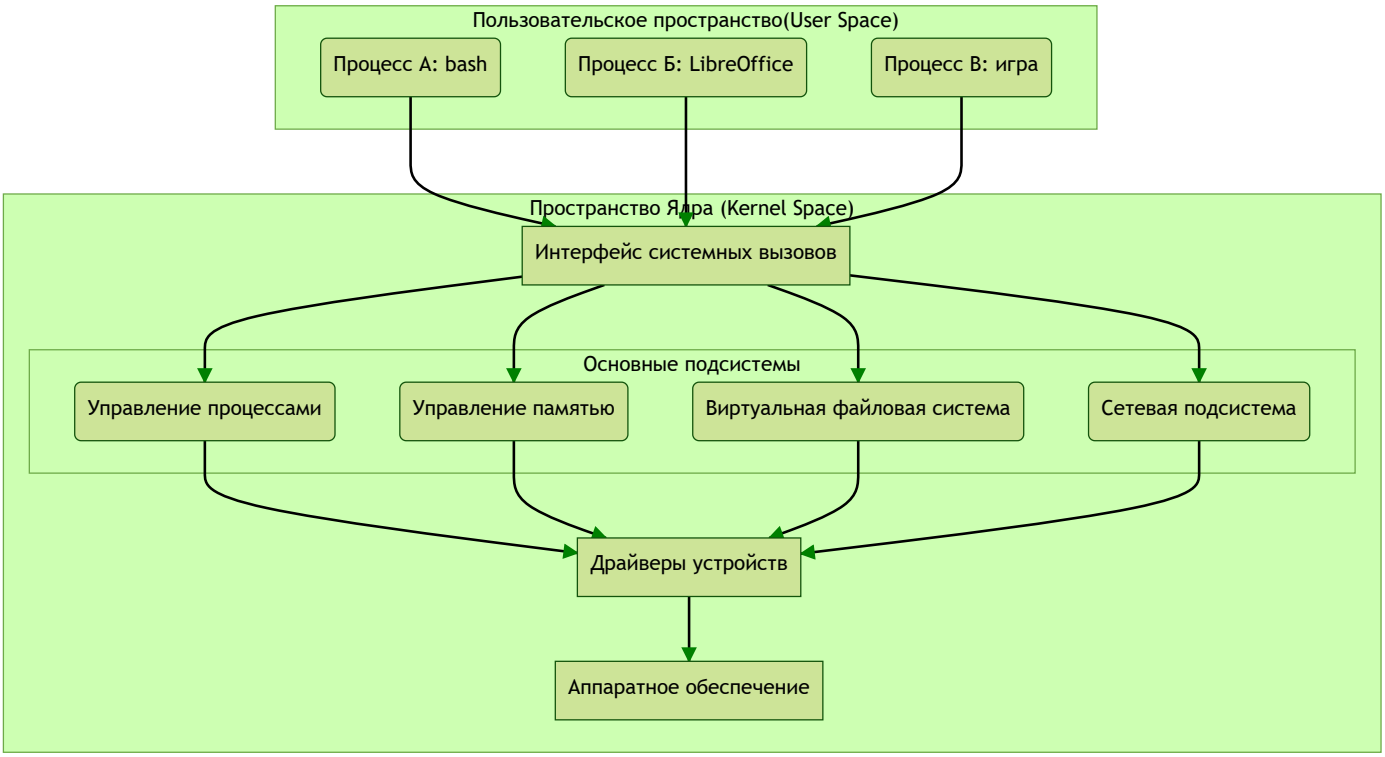
В Unix всё является файлом — обычные файлы, каталоги, устройства. Файловая система имеет иерархическую (древовидную) структуру с единым корнем `/`.

Ключевые концепции:

- **Inode (индексный дескриптор):** Это структура данных, в которой хранится вся метаинформация о файле:
  - Тип файла (обычный, каталог, символическая ссылка и т.д.).
  - Права доступа.
  - Владелец и группа.
  - Размер файла.
  - Временные метки (создание, модификация, доступ).
  - Указатели на блоки данных на диске, где хранится содержимое файла.
  - **Важно:** Inode не хранит имя файла.
- **Каталог (Directory):** Это специальный тип файла, который содержит список пар: **имя файла** ↔ **номер inode**. Таким образом, имя файла — это просто ссылка на inode.
- **Жесткая ссылка (Hard Link):** Это второе (или третье, и т.д.) имя для того же самого inode. Все жесткие ссылки на один файл абсолютно равноправны. Файл физически удаляется с диска только тогда, когда удаляется последняя жесткая ссылка на его inode.

- **Символическая (мягкая) ссылка (Symbolic Link / Soft Link):** Это специальный файл, который хранит не номер inode, а текстовый путь к другому файлу. При обращении к символической ссылке система перенаправляет запрос по этому пути. Если оригинальный файл удалить, ссылка станет “битой”.

Схема иерархической структуры ядра Unix-подобной системы.



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

Интерпретаторы команд (Shell)

Shell — это программа, которая предоставляет пользователю интерфейс для взаимодействия с ядром ОС. Он читает команды, введенные пользователем, и запускает соответствующие программы.

- **Примеры:** `sh` (Bourne shell), `bash` (Bourne-again shell, самый популярный), `csh`, `ksh`.
- **Функции:** Запуск программ, управление переменными окружения, перенаправление потоков, конвейеры (pipes), написание скриптов.

Стандартные потоки ввода-вывода

Для каждого процесса при запуске ОС автоматически открывает три стандартных потока:

1. **stdin (standard input, файловый дескриптор 0):** Стандартный поток ввода. По умолчанию связан с клавиатурой.
2. **stdout (standard output, файловый дескриптор 1):** Стандартный поток вывода. Сюда программа пишет свой “нормальный” результат. По умолчанию связан с терминалом (экраном).
3. **stderr (standard error, файловый дескриптор 2):** Стандартный поток ошибок. Сюда программа пишет сообщения об ошибках. По умолчанию также связан с терминалом.

Разделение вывода на `stdout` и `stderr` позволяет перенаправлять полезный результат в файл, при этом видя ошибки на экране.

Фильтры и перенаправление

- **Фильтр:** Это программа, которая читает данные из `stdin`, преобразует их и пишет результат в `stdout`. `grep`, `sort`, `wc`, `sed`, `awk` — классические примеры фильтров.
- **Перенаправление потоков:**
  - `> file`: Перенаправить `stdout` в `file` (файл будет перезаписан).
  - `>> file`: Добавить `stdout` в конец `file`.
  - `< file`: Взять `stdin` из `file` вместо клавиатуры.
  - `2> file`: Перенаправить `stderr` в `file`.
- **Конвейер (Pipe, `|`):** Самый мощный механизм. Он соединяет `stdout` одной команды с `stdin` следующей, позволяя строить цепочки команд-фильтров.
  - **Пример:** `cat messages.txt | grep "error" | sort` — эта команда сначала выводит содержимое файла `messages.txt` в `stdout`, который по конвейеру передается в `stdin` команды `grep`. `grep` отбирает строки со словом “error” и пишет их в свой `stdout`, который, в свою очередь, передается в `stdin` команды `sort`, которая сортирует строки и выводит финальный результат на терминал.

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

Основные команды

- `ls`: Показать содержимое каталога.
- `cd`: Сменить текущий каталог.
- `pwd`: Показать текущий рабочий каталог.
- `mkdir`: Создать каталог.
- `rmdir`: Удалить пустой каталог.
- `rm`: Удалить файл или каталог (`rm -r` для рекурсивного удаления).
- `cp`: Скопировать файл или каталог.
- `mv`: Переместить или переименовать файл/каталог.
- `cat`: Вывести содержимое файла.

- `touch`: Создать пустой файл или обновить время модификации существующего.
- `chmod`: Изменить права доступа к файлу.
- `grep`: Поиск по шаблону (регулярному выражению) в файле.
- `sort`: Сортировка строк.
- `head`/`tail`: Показать первые/последние строки файла.

Права доступа к файлам

Права определяют, кто и что может делать с файлом. Вывод `ls -la` показывает их в виде строки из 10 символов (например, `-rwxr-xr--`).

- **1-й символ:** Тип файла (`-` - обычный файл, `d` - каталог, `l` - символическая ссылка).
- **Символы 2-4:** Права для **владельца** файла (`u` - user).
- **Символы 5-7:** Права для **группы** (`g` - group).
- **Символы 8-10:** Права для **всех остальных** (`o` - others).

Каждая тройка состоит из прав: * `r` (read) - чтение. * `w` (write) - запись/модификация. * `x` (execute) - исполнение (для файлов) или вход (для каталогов).

Способы задания прав (команда `chmod`)

1. **Символьный (Symbolic):** Использует буквы `u, g, o, a` (all) и знаки `+, -, =`.
   - `chmod u+x file`: Добавить право на исполнение для владельца.
   - `chmod go-w file`: Убрать право на запись у группы и остальных.
   - `chmod a=r file`: Установить для всех только право на чтение.
2. **Восьмеричный (Octal):** Каждая тройка прав представляется одной восьмеричной цифрой.
   - `r`=4, `w`=2, `x`=1.
   - `rwx` = 4+2+1 = 7
   - `r-x` = 4+0+1 = 5
   - `rw-` = 4+2+0 = 6
   - **Пример:** `chmod 754 file` установит права `rwxr-xr--`.

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

Состав и структура БЭВМ

БЭВМ — это 16-разрядная ЭВМ аккумуляторного типа с Гарвардской архитектурой (хотя в лекциях упоминается и фон Нейман, но разделение памяти и I/O ближе к Гарвардской).

- **Процессор (CPU):**
  - **Регистры:**
    - `AC` (Аккумулятор, 16 бит): Основной рабочий регистр для арифметических операций.

    - `BR` (Буферный регистр, 16 бит): Вспомогательный регистр для временного хранения данных.
    - `PS` (Регистр состояния, 16 бит): Хранит флаги NZVC и управляющие биты.
    - `IP` (Счетчик команд, 11 бит): Хранит адрес следующей исполняемой команды.
    - `AR` (Регистр адреса, 11 бит): Хранит адрес ячейки памяти для текущего обращения.
    - `DR` (Регистр данных, 16 бит): Буфер для обмена данными с памятью.
    - `CR` (Регистр команд, 16 бит): Хранит код текущей исполняемой команды.
    - `SP` (Указатель стека, 11 бит): Хранит адрес вершины стека.
    - `IR` (Клавишный регистр, 16 бит): Регистр для ввода данных с пульта оператора.
  - **АЛУ и Коммутатор:** Выполняют операции и управляют потоками данных.
  - **Устройство управления (УУ/МПУ):** Интерпретирует команды и генерирует управляющие сигналы.
- **Память (RAM):** 2048 16-битных ячеек (слов).
- **Пульт оператора:** Средство для ручного взаимодействия с БЭВМ.

Адресные пространства

В БЭВМ существует два основных адресных пространства:

1. **Пространство памяти:** 2048 16-битных ячеек, адресуемых 11-битным адресом.
2. **Пространство ввода-вывода:** 256 8-битных регистров внешних устройств (контроллеров), адресуемых 8-битным номером регистра, который указывается в командах ввода-вывода.

Система и форматы команд

Команды 16-битные. Формат зависит от типа:

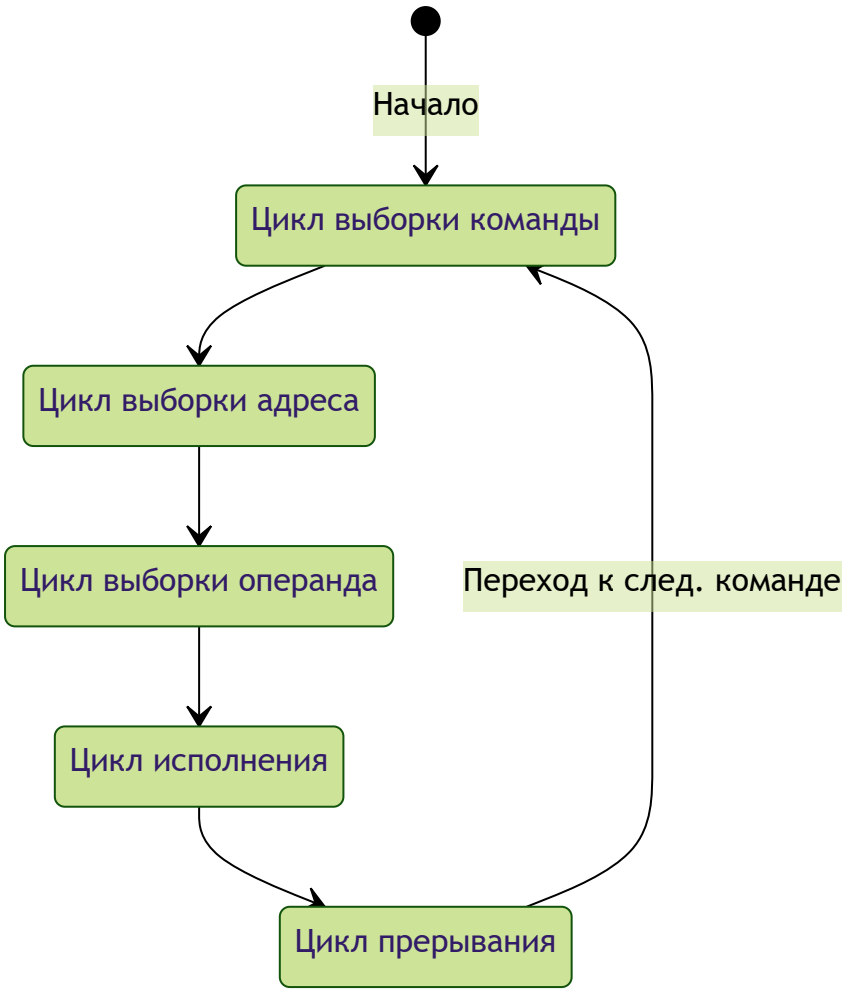
- **Безадресные:** Старшие 4 бита (КОП) = `0000`. Младшие биты — расширение кода операции.
- **Адресные:** Имеют сложный формат, включающий КОП, режим адресации и смещение/адрес.
- **Команды ветвления:** КОП = `1111`. Младшие биты — смещение для перехода.
- **Команды ввода-вывода:** КОП = `0001`. Часть битов кодирует приказ, часть — номер регистра ВУ.

Машинные циклы

Это последовательность шагов, которые УУ выполняет для исполнения одной команды.

1. **Цикл выборки команды (IF):** Загрузка кода команды из памяти (по адресу в IP) в регистр CR и инкремент IP.
2. **Цикл выборки адреса (AF):** Вычисление эффективного адреса операнда на основе режима адресации.
3. **Цикл выборки операнда (OF):** Загрузка операнда из памяти (по вычисленному адресу) в регистр DR.
4. **Цикл исполнения (EX):** Непосредственное выполнение операции, заданной командой.
5. **Цикл прерывания (INT):** Проверка наличия запросов на прерывание и, если есть, передача управления на обработчик прерывания.

Диаграмма состояний машинных циклов.



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

Организация вычислений

Все вычисления происходят с использованием аккумулятора (AC). Типичная последовательность:

1. Загрузить первый операнд из памяти в AC (`LD`).
2. Выполнить операцию (например, `ADD`), используя значение в AC и второй операнд из памяти. Результат помещается в AC.
3. Сохранить результат из AC в память (`ST`).

Сдвиги, арифметические и логические операции

- **Логические:** `AND`, `OR` (побитовые), `NOT` (инверсия AC).
- **Арифметические:** `ADD` (сложение), `ADC` (сложение с переносом, для длинной арифметики), `SUB` (вычитание), `INC`/`DEC` (инкремент/декремент AC), `NEG` (изменение знака, `NOT AC + 1`).
- **Сдвиги:**

- `ASL`/`ASR`: Арифметический сдвиг влево/вправо. `ASL` равносильно умножению на 2, `ASR` — делению на 2 со знаком.
- `ROL`/`ROR`: Циклический сдвиг через флаг переноса C. Все 17 бит (16 от AC и 1 от C) сдвигаются как кольцо.

Цикл выборки команды (IF) - подробно

Этот цикл состоит из трех микрокоманд (тактов):

1. **Такт 1:** `IP → BR, AR`. Адрес текущей команды из `IP` копируется в регистр адреса `AR` (для чтения из памяти) и в буферный регистр `BR` (для последующего инкремента).
2. **Такт 2:** `MEM(AR) → DR`, `BR + 1 → IP`. Одновременно происходит две операции: из памяти по адресу в `AR` считывается код команды в `DR`, и в `IP` записывается адрес *следующей* команды (`BR+1`).
3. **Такт 3:** `DR → CR`. Код команды из `DR` переписывается в регистр команд `CR` для дальнейшего декодирования устройством управления.

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

Организация массивов

Массив в БЭВМ — это просто непрерывный блок ячеек памяти. Программист сам отвечает за организацию доступа к элементам, обычно используя один из регистров в качестве указателя на текущий элемент массива.

Режимы адресации

Режим адресации определяет, как процессор находит операнд для команды.

- **Прямая абсолютная (`$L`):** В команде указан полный 11-битный адрес операнда. Цикл AF не нужен.
- **Прямая относительная (`L`):** В команде указано 8-битное смещение. Адрес операнда вычисляется как `IP + смещение`.
- **Косвенная (`(L)`):** Адрес, вычисленный как в прямом режиме, указывает не на операнд, а на ячейку, в которой хранится *адрес* операнда (указатель). Требует дополнительного обращения к памяти.
- **Автоинкрементная (`(L)+`):** Косвенный режим, при котором после использования указателя его значение в памяти автоматически увеличивается на 1. Идеально для последовательной обработки массивов.
- **Автодекрементная (`-(L)`):** Косвенный режим, при котором значение указателя сначала уменьшается на 1, а затем используется.
- **Непосредственная (`#N`):** Операнд является частью самой команды. Циклы AF и OF не нужны.

Циклы выборки адреса (AF) и операнда (OF)

Эти циклы выполняются после цикла IF для команд с относительной или косвенной адресацией.

- **Цикл AF:** Вычисляет эффективный адрес операнда. Для относительной адресации это `IP + смещение`. Результат помещается в `AR`.
- **Цикл OF:** Считывает данные по адресу из `AR`. Для прямой адресации — это сам операнд. Для косвенной — это адрес операнда, который снова помещается в `AR` для еще одного чтения.

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

Управление вычислительным процессом

Осуществляется с помощью **команд условного и безусловного переходов**, которые изменяют содержимое счетчика команд `IP`.

- **Безусловный переход (`JUMP`, `BR`):** Просто загружает в `IP` новый адрес.
- **Условный переход:** Изменяет `IP` только если выполнено некоторое условие, которое проверяется по флагам в регистре состояния `PS` (N, Z, V, C).
- `BEQ` (Branch if Equal): переход, если Z=1 (результат предыдущей операции был 0). * `BNE` (Branch if Not Equal): переход, если Z=0.
- `BMI` (Branch if Minus): переход, если N=1 (результат отрицательный). * `BPL` (Branch if Plus): переход, если N=0.
- `BCS`/`BCC`: переход по наличию/отсутствию переноса C (для беззнаковых сравнений).
- `BVS`/`BVC`: переход по наличию/отсутствию переполнения V.
- `BLT`/`BGE`: переход “меньше”/“больше или равно” (для знаковых сравнений, используют комбинацию флагов N и V).

Цикл исполнения команды LOOP

`LOOP` — это специализированная команда для организации циклов со счетчиком. Она объединяет три действия:

1. Уменьшает на единицу значение в ячейке памяти, на которую указывает.
2. Проверяет, стал ли результат меньше или равен нулю.
3. Если результат все еще положителен, выполняет **пропуск** следующей за `LOOP` команды (обычно это `JMP` или `BR` для выхода из цикла). Если результат <= 0, пропуск не выполняется, и программа выходит из цикла.

Цикл исполнения `LOOP`:

1. Выполняются циклы IF, AF, OF, чтобы загрузить значение счетчика в `DR`.
2. Содержимое `DR` уменьшается на 1. Микропрограмма для этого использует трюк `~0 + DR`, где `~0` это `0xFFFF` или `-1`.
3. Результат (`DR-1`) записывается обратно в память и одновременно в `BR`.
4. Проверяется знаковый бит `BR(15)`. Если он 0 (число положительное), то `IP` инкрементируется, пропуская следующую команду.
5. Если знаковый бит 1 (число <= 0), `IP` не меняется, и выполняется следующая команда (выход из цикла).

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

Подпрограммы и стек

- **Подпрограмма:** Именованный блок кода, который можно вызывать из разных мест основной программы для выполнения повторяющихся действий.
- **Вызов (`CALL`):** Команда `CALL` сохраняет адрес возврата (адрес следующей за `CALL` команды) в специальную область памяти — **стек** — и передает управление на начало подпрограммы.
- **Возврат (`RET`):** Команда `RET` извлекает адрес возврата с вершины стека и загружает его в счетчик команд `IP`, возвращая управление в основную программу.
- **Стек (Stack):** Область памяти, работающая по принципу LIFO (Last-In, First-Out). Управляется указателем стека `SP`.
  - `PUSH`: Поместить значение в стек. `SP` сначала уменьшается, затем по адресу в `SP` происходит запись.
  - `POP`: Извлечь значение из стека. Сначала происходит чтение по адресу в `SP`, затем `SP` увеличивается.

Передача параметров в подпрограмму

- **Через регистры:** Самый быстрый способ. В БЭВМ для этого можно использовать аккумулятор AC (можно передать до 16 однобитных флагов или одно 16-битное число) и флаг переноса C.
- **Через ячейки памяти:** Параметры размещаются в заранее известных ячейках памяти. Медленнее, но позволяет передать много данных.
- **Через стек:** Самый гибкий и распространенный способ. Перед вызовом `CALL` параметры помещаются в стек командами `PUSH`. Внутри подпрограммы к ним можно получить доступ, используя адресацию со смещением относительно `SP`.

Позиционно-независимый код (PIC)

Это код, который может быть загружен в любое место памяти и будет корректно работать.

- **Принцип:** Все внутренние переходы и обращения к данным внутри модуля используют **относительную адресацию** (относительно `IP`).
- **Внешние ссылки** (на библиотеки, ядро) должны быть абсолютными, и их адреса разрешаются на этапе загрузки.
- **В БЭВМ:** Наличие режимов относительной адресации позволяет писать позиционно-независимый код.

Загрузчик и библиотеки

- **Загрузчик (Loader):** Компонент ОС, который загружает исполняемый файл программы в память, настраивает ее адресное пространство и передает ей управление.
- **Линковщик (Linker):** Связывает код программы с кодом из библиотек.
- **Библиотеки:** Наборы готовых функций.

- **Статические:** Код библиотечных функций полностью копируется в исполняемый файл на этапе компиляции.
- **Динамические (разделяемые, `.so`, `.dll`):** Код библиотеки не копируется. В исполняемый файл помещаются только ссылки. Библиотека загружается в память один раз при запуске программы (или по требованию) и может использоваться несколькими программами одновременно. Это экономит память и позволяет обновлять библиотеки без перекомпиляции программ.

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

Общая организация ввода-вывода (I/O)

Взаимодействие CPU с внешними устройствами (ВУ) происходит не напрямую, а через **контроллеры**. * **Контроллер:** Специализированное электронное устройство, которое управляет работой одного или нескольких ВУ и предоставляет процессору стандартизированный интерфейс для обмена данными (набор регистров). * **Драйвер:** Программный модуль (часть ОС), который “знает”, как работать с регистрами конкретного контроллера, чтобы управлять устройством. Он предоставляет программам более высокий уровень абстракции (например, `read`, `write`).

Этапы обмена информацией

1. **Инициация обмена:** Процесс “договоренности” о начале обмена. Может быть:
   - **Синхронная:** Обмен начинается в заранее определенный момент времени (например, по таймеру).
   - **Асинхронная:** Обмен начинается по готовности. Процессор постоянно опрашивает флаг готовности устройства в цикле (**spin-loop**), что неэффективно тратит его время.
   - **Управляемая прерываниями:** Наиболее эффективный способ. Устройство само сигнализирует процессору о своей готовности, посылая аппаратный сигнал **прерывания**.
2. **Передача информации:** Непосредственный обмен данными между регистрами контроллера и памятью/регистрами процессора.
3. **Завершение обмена:** Приведение устройства и контроллера в исходное состояние для следующего обмена.

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

В БЭВМ I/O организован через **программно-управляемый ввод-вывод** с отдельным адресным пространством для регистров ВУ.

Устройства ввода-вывода (ВУ)

БЭВМ эмулирует ряд устройств, каждое из которых управляется своим контроллером (КВУ):

- **ВУ-0:** Таймер.

- **ВУ-1/2:** Простейшие устройства вывода/ввода с кнопкой “Готов”.
- **ВУ-3/4:** Комбинированные устройства ввода-вывода.
- **ВУ-5:** Текстовый принтер.
- **ВУ-6:** Бегущая строка.
- **ВУ-7:** 7-сегментный индикатор.
- **ВУ-8/9:** Клавиатура и цифровой NumPad.

Каждый контроллер имеет как минимум **регистр данных (РДВУ)** и **регистр состояния (РСВУ)**, в котором 6-й бит (`0x40`) обычно является флагом готовности.

Команды ввода-вывода

- `IN REG`: Чтение данных из регистра ВУ с номером `REG` в младший байт аккумулятора.
- `OUT REG`: Запись данных из младшего байта аккумулятора в регистр ВУ с номером `REG`.
- `DI` / `EI`: Запретить / Разрешить прерывания глобально.
- `IRET`: Возврат из обработчика прерывания.

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

Асинхронный обмен в БЭВМ (с опросом)

Это базовый способ обмена, когда процессор активно ждет готовности устройства.

Пример программы (ввод символа с ВУ-2):

```
WAIT:
    IN   5       ; Читаем регистр состояния SR#5 (ВУ-2)
    AND  #0x40   ; Оставляем только 6-й бит (готовность)
    BEQ  WAIT    ; Если бит 0 (не готов), повторяем цикл

    IN   4       ; Если готов, читаем данные из DR#4
```

Временные издержки: Основная проблема — **цикл ожидания (spin-loop)**. Пока процессор крутится в этом цикле, он не может выполнять никакой другой полезной работы. Если устройство медленное (например, пользователь долго не нажимает клавишу), процессорное время тратится впустую. Это крайне неэффективно.

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

Для решения проблемы spin-loop’ов используются прерывания.

- **Принцип:** ВУ, будучи готовым к обмену, само посылает процессору сигнал **запроса на прерывание (Interrupt Request, IntRq)**. Процессор, завершив текущую команду, приостанавливает основную программу и переходит к выполнению специальной подпрограммы — **обработчика прерывания**.
- **Вектор прерывания:** Чтобы процессор знал, на какую подпрограмму переходить для какого устройства, используется таблица векторов прерываний. Это заранее

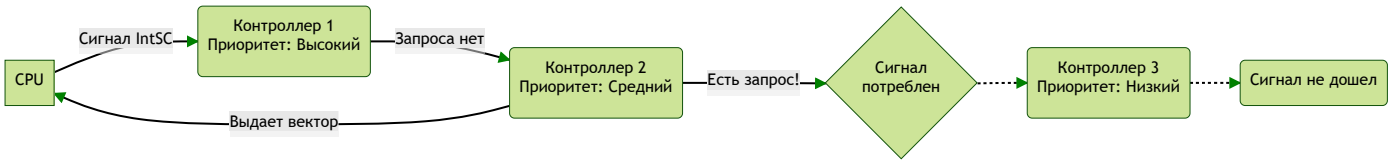
определенная область памяти (в БЭВМ — ячейки 0x0 - 0xF), где для каждого номера прерывания хранятся две величины:

1. Адрес подпрограммы-обработчика.
2. Новое значение регистра состояния (PS), с которым будет работать обработчик.

- **Контроллер прерывания:** Логика в контроллере ВУ, которая:

1. Формирует сигнал `IntRq` при наступлении события (например, нажатие кнопки “Готов”).
2. Хранит номер вектора прерывания, который он должен выдать.
3. Участвует в определении приоритета. В БЭВМ используется **цепочечная схема (daisy-chain)**: сигнал предоставления прерывания идет от процессора последовательно через все контроллеры. Первый в цепочке, кто запросил прерывание, “перехватывает” этот сигнал и не пропускает его дальше, тем самым получая наивысший приоритет.

Схема распространения сигнала предоставления прерывания.



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

Пример программы с прерываниями

Программа состоит из двух частей:

1. **Блок инициализации:**

- Заполнить таблицу векторов прерываний адресами обработчиков (`ORG 0x0 ...`).
- Запретить прерывания (`DI`). * Настроить контроллеры: записать в их регистры управления (MR) номер вектора и бит разрешения прерывания для данного контроллера.
- Разрешить прерывания глобально (`EI`).

1. **Основная программа:** Выполняет свою работу, не обращая внимания на ввод-вывод.
2. **Обработчики прерываний:** Подпрограммы, которые выполняют обмен с ВУ. Типичный обработчик:

- Сохранить состояние (например, `PUSH AC`), т.к. основная программа могла использовать аккумулятор.
- Выполнить обмен данными с ВУ (`IN`/`OUT`).
- Восстановить состояние (`POP AC`).
- Вернуться из прерывания (`IRET`).

Цикл прерывания (INT)

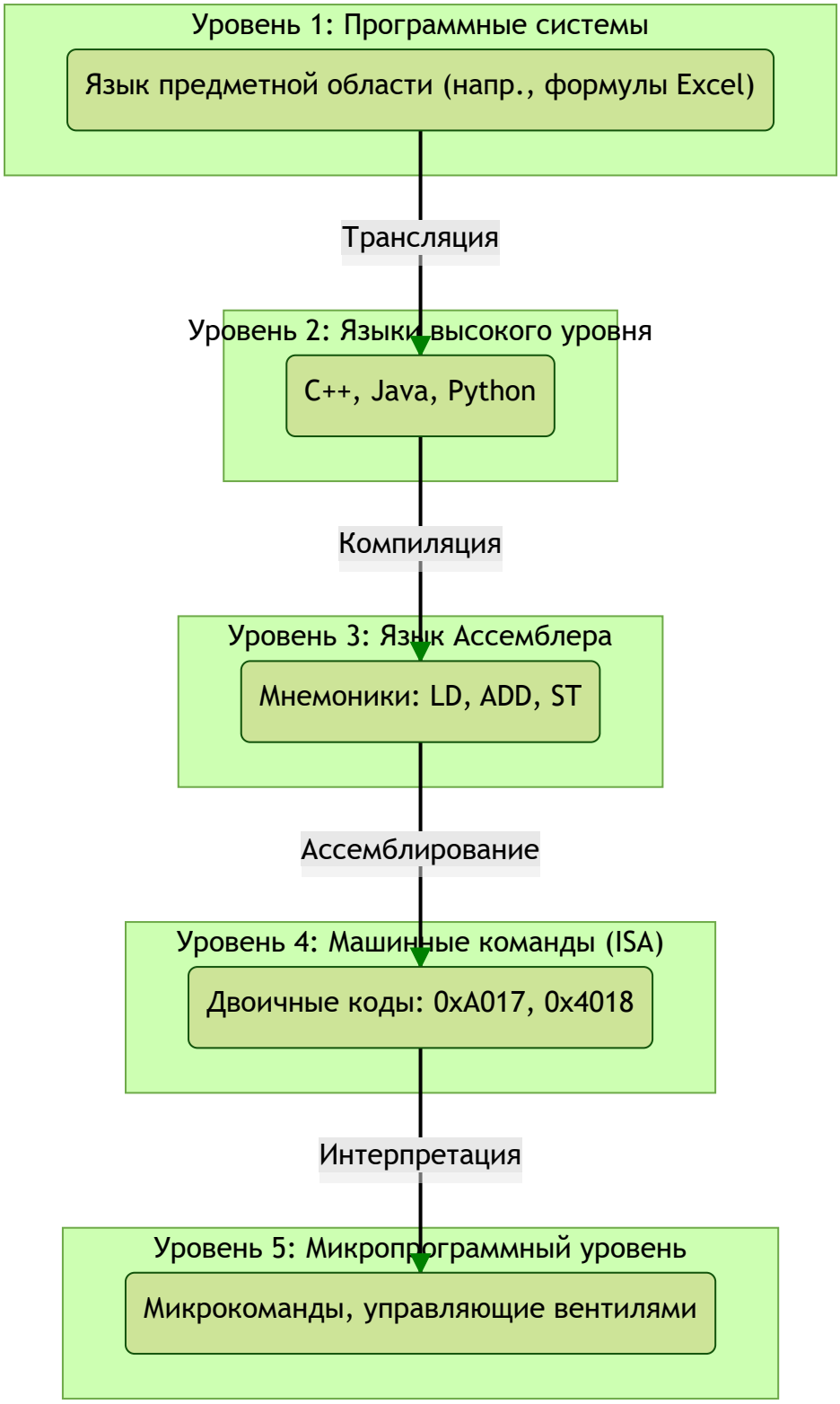
Это машинный цикл, который выполняется, если обнаружен активный запрос на прерывание.

1. Проверяются флаги (работа/останов, разрешены ли прерывания).
2. Формируется сигнал предоставления прерывания (`IntSC`).
3. Процессор ожидает от контроллера номер вектора на шине.
4. Текущие `IP` и `PS` сохраняются в стеке.
5. Из таблицы векторов по полученному номеру загружаются новые `IP` (адрес обработчика) и `PS`.
6. Происходит переход к выполнению обработчика прерывания.

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

Концепция многоуровневой ЭВМ заключается в том, что современный компьютер можно рассматривать как иерархию виртуальных машин. Каждый уровень этой иерархии предоставляет определенный набор инструкций и объектов, скрывая сложность нижележащих уровней. Программы, написанные на одном уровне, транслируются (компилируются) или интерпретируются в команды нижележащего уровня.

Иерархия уровней абстракции в ЭВМ.



Уровни (сверху вниз):

1. **Уровень программных систем (специальный язык):** Самый близкий к человеку. Пользователь взаимодействует с компьютером в терминах предметной области, а не программирования.
   - **Пример:** Ввод формулы `=A1+B1` в Excel. Человек оперирует ячейками и математическими операциями, а не переменными и инструкциями.
2. **Уровень проблемно-ориентированных (алгоритмических) языков:** Уровень языков высокого уровня (C++, Java, Python). Программист описывает алгоритм решения задачи.
   - **Пример программы (`=A1+B1`):**

```
int a = read_cell("A1");
int b = read_cell("B1");
int result = a + b;
write_cell("C1", result);
```

3. **Уровень языка Ассемблера:** Низкоуровневый язык, где команды напрямую соответствуют машинным инструкциям процессора. Вместо сложных конструкций используются простые мнемоники команд и символические имена для адресов (метки).
   - **Пример программы (фрагмент):**

```
LD   A1_ADDR   ; Загрузить значение из ячейки A1 в аккумулятор
ADD  B1_ADDR   ; Сложить со значением из ячейки B1
ST   C1_ADDR   ; Сохранить результат в ячейку C1
```

4. **Уровень машинных команд (архитектура набора команд, ISA):** Уровень, который “понимает” процессор. Программы представлены в виде последовательности двоичных кодов команд.
   - **Пример программы:** `0xA017` (LD 0x17) `0x4018` (ADD 0x18) `0xE019` (ST 0x19)
5. **Микропрограммный уровень:** Внутренний уровень устройства управления. Каждая машинная команда исполняется как последовательность еще более простых шагов — **микрокоманд**. Каждая микрокоманда соответствует открытию/закрытию определенных вентилей, управляющих потоками данных между регистрами и АЛУ.
   - **Пример (часть исполнения `ADD`):**
     - Микрокоманда 1: Открыть вентили `RDAC` и `RDDR` (подать AC и DR на АЛУ).
     - Микрокоманда 2: Открыть вентиль `WRAC` (записать результат из АЛУ в AC).

Эта многоуровневая структура позволяет абстрагироваться от сложности аппаратуры и разрабатывать программы на удобном уровне.

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

Структура Микропрограммного Устройства Управления (МПУ)

МПУ в БЭВМ — это, по сути, “процессор внутри процессора”, который исполняет не машинные команды, а микрокоманды. Его задача — интерпретировать код машинной команды (из регистра `CR`) и сгенерировать правильную последовательность управляющих сигналов (открытия/закрытия вентилей) для ее выполнения.

- **Память микрокоманд:** ПЗУ, в котором хранится микропрограмма (интерпретатор). В БЭВМ это 256 ячеек по 40 бит.
- **Счетчик микрокоманд (СчМК):** Аналог `IP`, указывает на адрес следующей микрокоманды.
- **Регистр микрокоманд (РМК):** Аналог `CR`, хранит текущую исполняемую микрокоманду.
- **Схема управления:** Декодирует части микрокоманды и формирует управляющие сигналы.

Форматы микрокоманд в БЭВМ

В БЭВМ используется **горизонтальное микропрограммирование**: каждый бит в микрокоманде (или небольшая группа битов) напрямую управляет своим вентилем или блоком. Это обеспечивает максимальный параллелизм, но приводит к длинным микрокомандам (40 бит). Существует два типа микрокоманд, которые различаются по 39-му биту:

1. **Операционная микрокоманда (ОМК, бит 39 = 0):**
   - **Назначение:** Выполнение операций с данными — пересылки между регистрами, операции в АЛУ, работа с памятью.
   - **Структура:** Представляет собой набор полей (групп битов), каждое из которых управляет своим блоком:
     - Биты 0-7: Чтение из регистров (`RDAC`, `RDDR` и т.д.).
     - Биты 8-11: Управление АЛУ (`COML`, `COMR`, `PLS1`, `SORA`).
     - Биты 12-23: Управление коммутатором и установкой флагов.
     - Биты 24-31: Запись в регистры (`WRAC`, `WRBR` и т.д.).
     - Биты 32-35: Управление памятью и вводом-выводом.
     - Бит 38: `HALT` (останов).
2. **Управляющая микрокоманда (УМК, бит 39 = 1):**
   - **Назначение:** Анализ состояния регистров и выполнение условных/безусловных переходов внутри микропрограммы. Это необходимо для декодирования машинных команд.
   - **Структура:**
     - Часть полей совпадает с ОМК (чтение из регистров, управление АЛУ).
     - Поле **ADDR** (24-31): Адрес перехода внутри микропрограммы.
     - Поле **BIT** (16-23): Выбор бита для проверки.
     - Поле **COMP** (32): С чем сравнивать выбранный бит (с 0 или 1).
     - **Принцип работы:** Если `(значение_выбранного_бита) == COMP`, то в СчМК загружается адрес из поля ADDR (происходит переход). Иначе — выполняется следующая по порядку микрокоманда.

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

Арифметико-логическое устройство (АЛУ)

АЛУ выполняет все арифметические и логические операции.

- **Структура:** * Два 16-битных входа: левый и правый.
- **Инверторы на входах:** Позволяют инвертировать (получить обратный код) каждый из операндов по отдельности. Управляются сигналами `COML` и `COMR`.
- **Основной блок:** Состоит из двух параллельно работающих схем:
  1. 16-разрядный **сумматор**, который может складывать операнды и добавлять единицу (сигнал `PLS1`). Используется для сложения, вычитания (`A + ~B + 1`), инкремента (`A + 0 + 1`) и т.д.

2. Схема **логического умножения (И)**. * **Выбор операции:** Сигнал `SORA` (Sum OR And) выбирает, результат какой схемы (суммы или И) пойдет на выход.

- **Принцип работы:** На входы АЛУ подаются операнды. Управляющие сигналы (`COML`, `COMR`, `PLS1`, `SORA`) настраивают АЛУ на нужную операцию. Результат операции поступает на коммутатор.

Коммутатор

Коммутатор — это блок, который выполняет сдвиги и перестановку байт в слове, полученном от АЛУ.

- **Операции:**
  - **Прямая передача:** `LTOL` (Low to Low), `HTOH` (High to High) — передача младшего/старшего байта без изменений.
  - **Обмен байт:** `LTOH`, `HTOL` — младший и старший байты меняются местами (аналог команды `SWAB`).
  - **Сдвиги:** `SHLT`/`SHRT` (арифметические), `SHLO`/`SHRF` (циклические).
  - **Расширение знака (`SEXT`):** 7-й бит (знак младшего байта) копируется во все биты старшего байта.

Регистр состояния (PS)

Хранит флаги и управляющие биты.

- **Флаги NZVC (биты 0-3):** Отражают результат последней операции. Устанавливаются сигналами `SETC`, `SETV`, `STNZ`.
- **Управляющие биты:**
  - **EI (бит 5):** Глобальное разрешение прерываний.
  - **INT (бит 6):** Флаг запроса на прерывание от устройства.
  - **W (бит 7):** Состояние тумблера РАБОТА/ОСТАНОВ.
  - **P (бит 8):** Флаг работы программы (влияет на тактовый генератор).

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

Микропрограммное управление

Это процесс управления потоками данных в процессоре путем последовательной подачи сигналов на управляющие входы вентилей. Эти последовательности сигналов и есть микропрограмма.

Пример микрооперации `DR` → `AC`: Чтобы переслать данные из регистра данных (`DR`) в аккумулятор (`AC`), МПУ должно в одном такте выполнить следующую **микрокоманду**:

1. Установить в 1 бит, соответствующий вентилю чтения из `DR` (`RDDR`).
2. Установить в 1 бит, соответствующий вентилю записи в `AC` (`WRAC`).
3. Все остальные биты, управляющие вентилями, установить в 0.

В результате данные с выхода `DR` по шине поступят на вход `AC`, и по сигналу записи `WRAC` запишутся в аккумулятор.

Интерпретатор БЭВМ

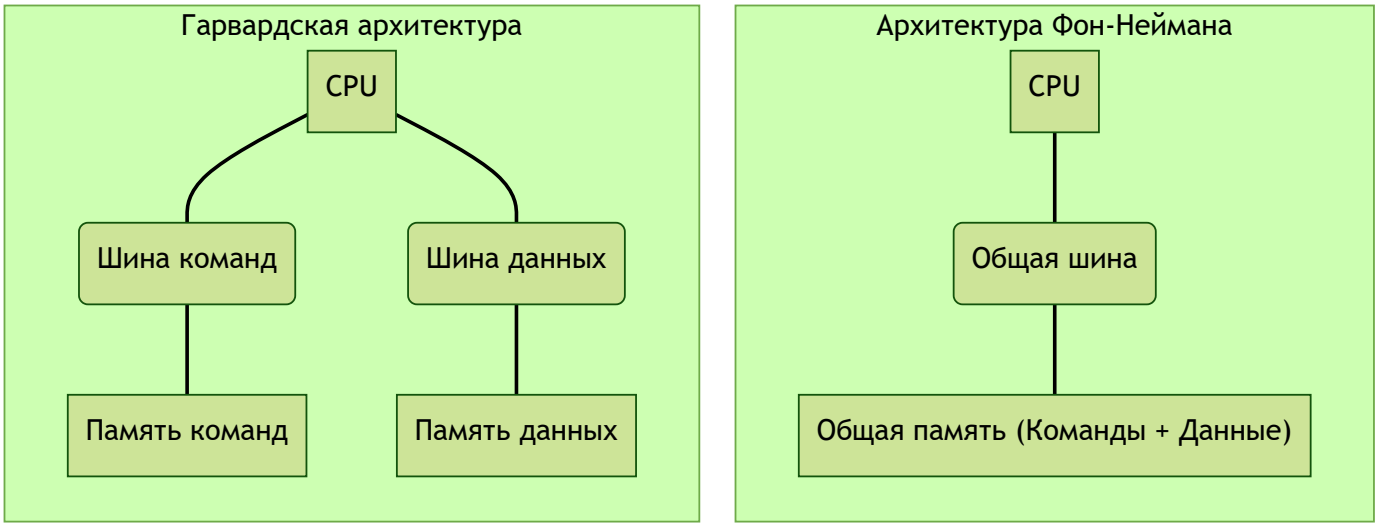
Это полная микропрограмма, хранящаяся в памяти МПУ. Он реализует логику всех машинных циклов (IF, AF, OF, EX, INT).

- **Структура:** Интерпретатор представляет собой большую таблицу, где каждой машинной команде соответствует своя последовательность (ветка) микрокоманд.
- **Принцип работы:**
  1. Начинается с цикла выборки команды (IF), который одинаков для всех.
  2. После того как код команды оказывается в `CR`, начинается **декодирование**. Это серия условных переходов (УМК), которые проверяют биты в `CR` и направляют выполнение по нужной ветке микропрограммы.
  3. Выполняется ветка, соответствующая данной команде (циклы AF, OF, EX). 4. В конце каждой ветки происходит безусловный переход на цикл прерывания (INT), а оттуда — обратно на начало, к циклу выборки следующей команды.

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

- **Архитектура фон Неймана:**
  - **Основной принцип: Принцип хранимой программы**. И команды, и данные хранятся в **единой общей памяти**.
  - **Обмен:** Процессор обменивается с памятью по **общей системной шине**.
  - **Недостаток:** “Бутылочное горлышко фон Неймана” — общая шина ограничивает производительность, так как процессор не может одновременно читать команду и данные.
  - **Применение:** Большинство современных компьютеров общего назначения (десктопы, серверы).
- **Гарвардская архитектура:**
  - **Основной принцип: Физическое разделение памяти** на память команд и память данных.
  - **Обмен:** Используются **две независимые шины** для доступа к каждой из памятей.
  - **Преимущество:** Процессор может одновременно выбирать следующую команду и работать с данными из предыдущей, что значительно повышает производительность (основа конвейерной обработки).
  - **Применение:** Цифровые сигнальные процессоры (DSP), микроконтроллеры, кэш-память современных процессоров (где кэш инструкций L1 и кэш данных L1 разделены).

БЭВМ имеет черты Гарвардской архитектуры, так как у нее разделены адресные пространства для памяти и для устройств ввода-вывода, что позволяет вести обмен с ними параллельно.

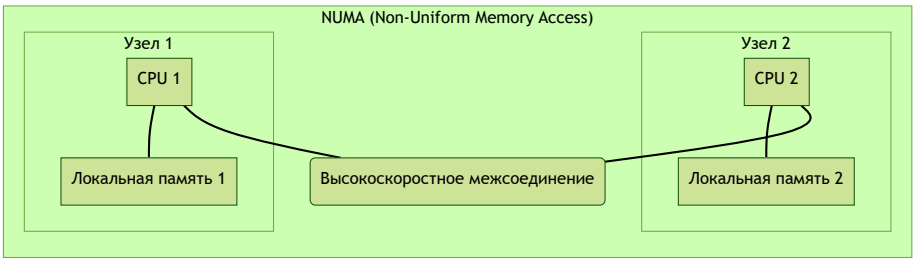


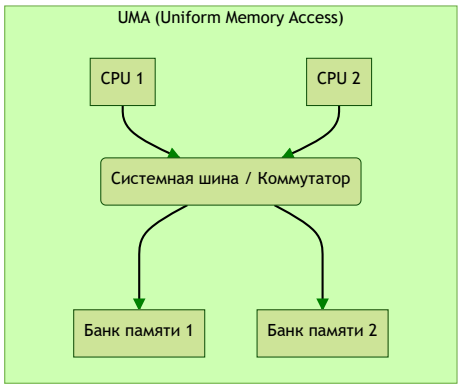
24. Архитектура многопроцессорных ЭВМ. Системный коммутатор. Архитектуры UMA и NUMA.

Многопроцессорные системы

Системы, содержащие несколько центральных процессоров (CPU).

- **UMA (Uniform Memory Access - Равномерный доступ к памяти):**
  - **Принцип:** Все процессоры имеют одинаковый по времени и логике доступ ко всей общей оперативной памяти. Память логически является единым целым.
  - **Реализация:** Процессоры и память подключаются к общей системной шине или к более сложному устройству — **системному коммутатору (Crossbar Switch)**. Коммутатор позволяет устанавливать одновременные соединения между несколькими парами “процессор-память”, в отличие от шины, которая в один момент времени может обслуживать только одну передачу.
  - **Применение:** Системы с небольшим числом процессоров (типичные многоядерные десктопы).
- **NUMA (Non-Uniform Memory Access - Неравномерный доступ к памяти):**
  - **Принцип:** Память физически распределена по узлам (node). Каждый узел содержит один или несколько процессоров и свой банк “локальной” памяти.
  - **Доступ:** Процессор имеет очень быстрый доступ к своей локальной памяти и более медленный — к памяти других узлов (удаленной памяти) через высокоскоростное межсоединение (например, Crossbar).
  - **Преимущество:** Отличная масштабируемость. Позволяет строить системы с сотнями и тысячами процессоров.
  - **Недостаток:** Сложность для программиста и операционной системы, которые должны стараться размещать данные процесса в локальной памяти того узла, где он исполняется, чтобы минимизировать медленные обращения к удаленной памяти.
  - **Применение:** Высокопроизводительные серверы и суперкомпьютеры.





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

Структура и окружение

Современный процессор — это сложнейшая система на кристалле, включающая:

- Несколько **ядер**, каждое из которых является полноценным CPU.
- Многоуровневый **кэш** (`L1`, `L2`, `L3`).
- **Контроллер памяти** для прямого взаимодействия с ОЗУ.
- **Контроллер шины** (например, PCIe) для связи с периферийными устройствами.
- Иногда **встроенное графическое ядро**.
- **MMU** (блок управления памятью).

Классификация архитектур набора команд (ISA)

- **CISC (Complex Instruction Set Computer):**
  - **Идея:** Иметь большой набор мощных, сложных команд, каждая из которых выполняет несколько низкоуровневых действий (например, `LOOP` в БЭВМ). Цель — сократить разрыв между языками высокого уровня и машинными командами.
  - **Характеристики:** Много режимов адресации, команды переменной длины.
  - **Проблема:** Сложность декодирования команд мешает эффективной конвейеризации.
  - **Пример:** Архитектура Intel x86. (Хотя современные x86 процессоры внутри являются RISC-подобными: они декодируют сложные CISC-команды в последовательность простых внутренних микроопераций).
- **RISC (Reduced Instruction Set Computer):**
  - **Идея:** Противоположная CISC. Набор команд должен быть маленьким, простым и фиксированной длины. Каждая команда выполняется за один такт.

- **Характеристики:** Большое количество регистров общего назначения, операции типа “load-store” (арифметические операции работают только с регистрами, для работы с памятью есть отдельные команды загрузки/сохранения).
- **Преимущество:** Простота декодирования и идеальная приспособленность для конвейерной обработки, что дает высокую производительность.
- **Примеры:** ARM, SPARC, MIPS, RISC-V.

- **VLIW (Very Long Instruction Word):**
  - **Идея:** Упаковать в одну очень длинную машинную команду несколько независимых простых операций, которые могут быть выполнены параллельно на разных исполнительных устройствах процессора.
  - **Отличие от суперскалярных RISC:** В RISC процессор сам находит независимые инструкции для параллельного выполнения. В VLIW эта задача полностью ложится на **компилятор**. Компилятор анализирует код и формирует эти “пакеты” операций.
  - **Преимущество:** Упрощение аппаратной части процессора.
  - **Недостаток:** Сложность компилятора, сильная зависимость производительности от его качества.
  - **Пример:** Процессоры “Эльбрус”.

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

Организация адресуемой памяти

Адресуемая память — это тип памяти, где каждая ячейка (или слово) имеет уникальный числовой адрес. Процессор получает доступ к данным, выставляя нужный адрес на шину адреса.

Структура (на примере БЭВМ):

1. **Дешифратор адреса:** Это основная схема, которая преобразует двоичный адрес, пришедший из регистра адреса (`AR`), в сигнал на одной-единственной линии, которая активирует нужную строку ячеек памяти.
2. **Массив ячеек памяти:** Двумерный массив элементов хранения (в БЭВМ 2048 строк по 16 бит).
3. **Усилители и мультиплексоры:** При чтении усиливают слабый сигнал от ячеек памяти и направляют его на шину данных. При записи направляют данные с шины в выбранную строку ячеек.

Временные диаграммы

Диаграммы показывают, как сигналы на шинах (адреса, данных, управления) изменяются во времени в течение одного цикла обмена с памятью.

- **Цикл чтения:**
  1. Процессор выставляет адрес на **шину адреса**.
  2. Процессор выставляет сигнал “Чтение” на **шину управления**.
  3. Через некоторое время (**время доступа, Тд**), память выставляет считанные данные на **шину данных**.

4. Память выставляет сигнал “Готовность”, сообщая процессору, что данные на шине валидны.
5. Процессор считывает данные и снимает сигналы. Памяти нужно **время восстановления (Тв)** перед следующим циклом.

- **Цикл записи:**

1. Процессор выставляет адрес на **шину адреса**.
2. Процессор выставляет данные для записи на **шину данных**.
3. Процессор выставляет сигнал “Запись” на **шину управления**.
4. Память записывает данные и выставляет сигнал “Готовность”.
5. Процессор снимает сигналы.

Конструктивные особенности современной памяти (DDR SDRAM)

- **Синхронная работа:** Обмен данными синхронизирован с тактовым генератором системной шины, что позволяет точно прогнозировать время ответа.
- **DDR (Double Data Rate):** Данные передаются дважды за один такт — и по переднему фронту (нарастанию) сигнала, и по заднему (спаду), что эффективно удваивает скорость передачи.
- **Burst Mode (Пакетный режим):** Вместо того чтобы передавать адрес для каждого слова, передается только начальный адрес, а затем несколько слов подряд (пакет) передаются на высокой скорости. Это эффективно для кэш-памяти, которая всегда запрашивает целую строку (cache line).
- **Interleaving (Расслоение памяти):** Память физически разделена на несколько независимых банков. Пока один банк завершает операцию, процессор может начать операцию с другим банком, что скрывает задержки и повышает общую пропускную способность.
- **SPD (Serial Presence Detect):** Небольшой чип ПЗУ на модуле памяти, который хранит информацию о характеристиках модуля (частота, тайминги, объем). BIOS считывает эти данные при старте компьютера для автоматической настройки.

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

Этот тип памяти характеризуется тем, что доступ к данным происходит не побайтно, а большими блоками (записями). Это связано с механической природой таких устройств.

Дисковая память (Жесткие диски, HDD)

- **Структура:** Состоит из нескольких вращающихся магнитных дисков (“блинов”). Данные записываются на концентрические **дорожки (tracks)**. Каждая дорожка разбита на **секторы** (обычно по 512 байт или 4 КБ). Совокупность дорожек с одинаковым номером на всех поверхностях называется **цилиндром**.
- **Доступ:** Осуществляется с помощью блока магнитных головок, который перемещается над поверхностью дисков. Время доступа складывается из трех компонентов:

1. **Время позиционирования (Тпоз):** Время перемещения головок к нужному цилиндру. Самая медленная часть.
2. **Время вращательного ожидания (Твр):** Время ожидания, пока нужный сектор подойдет под головку. В среднем — половина оборота диска.
3. **Время передачи:** Непосредственно чтение/запись данных сектора.

- **Особенности:** Прямой (блочный) доступ. Можно обратиться к любому блоку по его адресу (цилиндр, головка, сектор), но это медленно из-за механики.

Память на магнитных лентах (стримеры)

- **Структура:** Магнитная лента, намотанная на катушки.
- **Доступ: Последовательный (sequential)**. Чтобы прочитать данные в середине ленты, нужно перемотать всю ленту с начала до этого места.
- **Применение:** Из-за очень медленного доступа используется в основном для долгосрочного архивного хранения и резервного копирования больших объемов данных, где важна стоимость хранения и надежность, а не скорость доступа.

28. Характеристики запоминающих устройств. Пирамида памяти.

Основные характеристики

- **Месторасположение:** Процессорные (регистры), внутренние (ОЗУ, кэш), внешние (диски, ленты).
- **Емкость:** Объем данных, который может хранить устройство (байты, килобайты, гигабайты и т.д.).
- **Единица пересылки:** Минимальный объем данных, которым можно обменяться за одну операцию (слово для ОЗУ, строка для кэша, блок/сектор для диска).
- **Метод доступа:** Произвольный (RAM), прямой (диск), последовательный (лента), ассоциативный (кэш).
- **Быстродействие:** Определяется **временем доступа** (от запроса до получения первого бита) и **скоростью передачи**.
- **Физический тип:** Полупроводниковая (SRAM, DRAM), магнитная (диски, ленты), оптическая.
- **Стоимость:** Цена за бит хранения.

Пирамида памяти

Это иерархическая модель организации памяти в компьютере, которая отражает компромисс между скоростью, объемом и стоимостью.

- **Вершина пирамиды:**
  - **Устройства:** Регистры процессора, кэш-память (L1, L2, L3).
  - **Характеристики:** Очень высокая скорость, малый объем, очень высокая стоимость.
- **Середина пирамиды:**
  - **Устройства:** Основная память (ОЗУ), твердотельные накопители (SSD).
  - **Характеристики:** Средняя скорость, большой объем, средняя стоимость.
- **Основание пирамиды:**
  - **Устройства:** Жесткие диски (HDD), магнитные ленты.
  - **Характеристики:** Низкая скорость, огромный объем, очень низкая стоимость.

Принцип работы: Часто используемые данные и команды автоматически копируются с медленных нижних уровней на быстрые верхние (например, из ОЗУ в кэш). Когда процессору нужны данные, он сначала ищет их на самом быстром уровне (L1 кэш). Если

находит (**попадание, hit**), доступ происходит очень быстро. Если не находит (**промах, miss**), он ищет на следующем, более медленном уровне, и так далее. Эффективность такой системы основана на **принципе локальности ссылок**: если программа обратилась к ячейке памяти, то с большой вероятностью она скоро обратится к ней или к соседним ячейкам снова.

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

Ассоциативная память

Это особый тип памяти, доступ к которой осуществляется не по адресу, а **по содержимому**. При поиске в ассоциативную память подается искомое значение (ключ), и память мгновенно возвращает связанные с ним данные, если такой ключ найден. Она аппаратно очень сложна и дорога, поэтому используется для небольших объемов, например, в TLB и для хранения тегов в кэш-памяти.

Кэш-память (Cache)

Это небольшая, но очень быстрая статическая память (SRAM), расположенная между процессором и основной (более медленной) оперативной памятью (DRAM).

- **Принцип работы:** Кэш хранит копии наиболее часто используемых блоков данных из ОЗУ.
- **Структура:** Память кэша разбита на строки (cache lines). Каждой строке соответствует **тег**, который хранится в ассоциативной памяти тегов. Тег — это часть адреса, которая однозначно идентифицирует, какой именно блок из ОЗУ сейчас находится в данной строке кэша.
- **Процесс доступа:**

1. Процессор запрашивает данные по адресу из ОЗУ.
2. Контроллер кэша разбивает этот адрес на тег и номер строки.
3. Он одновременно ищет тег в ассоциативной памяти тегов.
4. **Попадание (hit):** Если тег найден, данные быстро считываются из соответствующей строки кэша и передаются процессору.
5. **Промах (miss):** Если тег не найден, происходит задержка. Процессор останавливается, а контроллер кэша загружает нужный блок данных из медленной ОЗУ в одну из строк кэша (возможно, вытесняя старые данные) и обновляет тег. После этого данные передаются процессору.

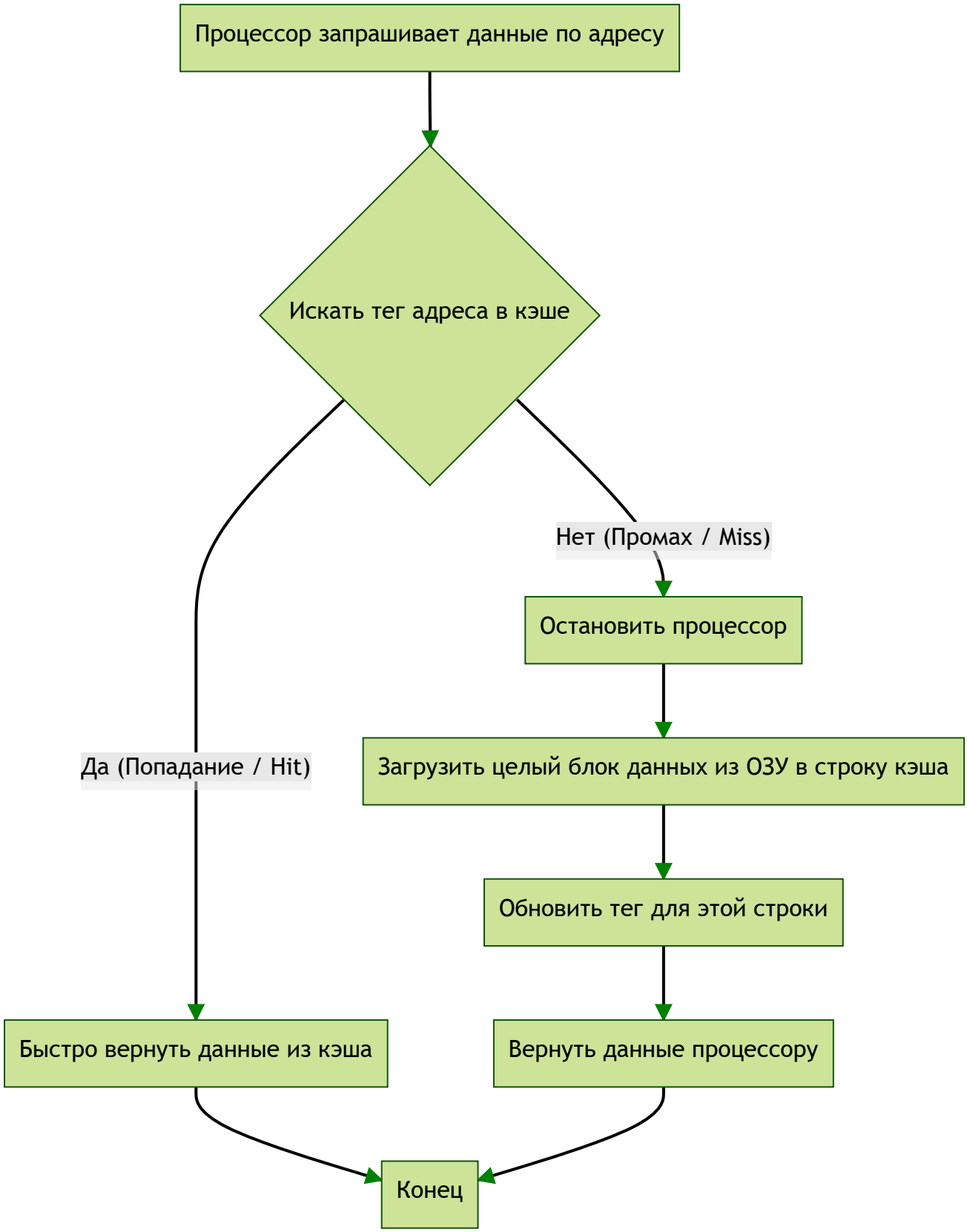
Влияние промахов на производительность

Производительность системы сильно зависит от **доли попаданий (hit rate)**.

- **Высокий hit rate (>95%):** Большинство обращений обслуживается быстрым кэшем, и среднее время доступа к памяти близко ко времени доступа к кэшу. Система работает быстро.
- **Низкий hit rate:** Часто происходят промахи. Каждое обращение к памяти приводит к долгому циклу загрузки данных из ОЗУ, что сводит на нет преимущество от наличия кэша. Процессор простаивает, производительность резко падает. Как видно на графике

из лекции, падение производительности нелинейное: снижение попаданий с 98% до 96% уже существенно замедляет систему, а при 86% попаданий производительность падает в несколько раз.

Флоу-чарт процесса обращения к кэш-памяти.



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

трансляции (TLB).

Предназначение виртуальной памяти

Виртуальная память — это механизм, который:

1. **Изолирует адресные пространства процессов:** Каждая программа работает в своем собственном виртуальном адресном пространстве (например, от 0 до 2^{64}-1). Она “думает”, что владеет всей памятью компьютера. Это защищает процессы друг от друга и от операционной системы.
2. **Позволяет использовать больше памяти, чем физически установлено:** Ненужные в данный момент части программы могут быть выгружены на диск в специальную **область подкачки (swap)**, освобождая физическую память для других задач.
3. **Упрощает управление памятью:** ОС работает с единым, непрерывным виртуальным пространством для каждого процесса, не заботясь о фрагментации физической памяти.

Сегментно-страничная организация

Это доминирующий способ организации виртуальной памяти.

- **Сегмент:** Логическая область памяти программы (сегмент кода, сегмент данных, сегмент стека).
- **Страница (Page):** Виртуальное адресное пространство делится на блоки фиксированного размера (например, 4 КБ), называемые страницами.
- **Страничный кадр (Page Frame):** Физическая память (ОЗУ) также делится на блоки такого же размера.

Принцип: ОС ведет для каждого процесса **таблицы страниц**, которые устанавливают соответствие между виртуальными страницами и физическими кадрами. Виртуальный адрес состоит из двух частей: **номера виртуальной страницы** и **смещения внутри страницы**.

Устройство управления памятью (MMU)

MMU — это аппаратный блок процессора, который на лету выполняет **трансляцию** виртуального адреса в физический.

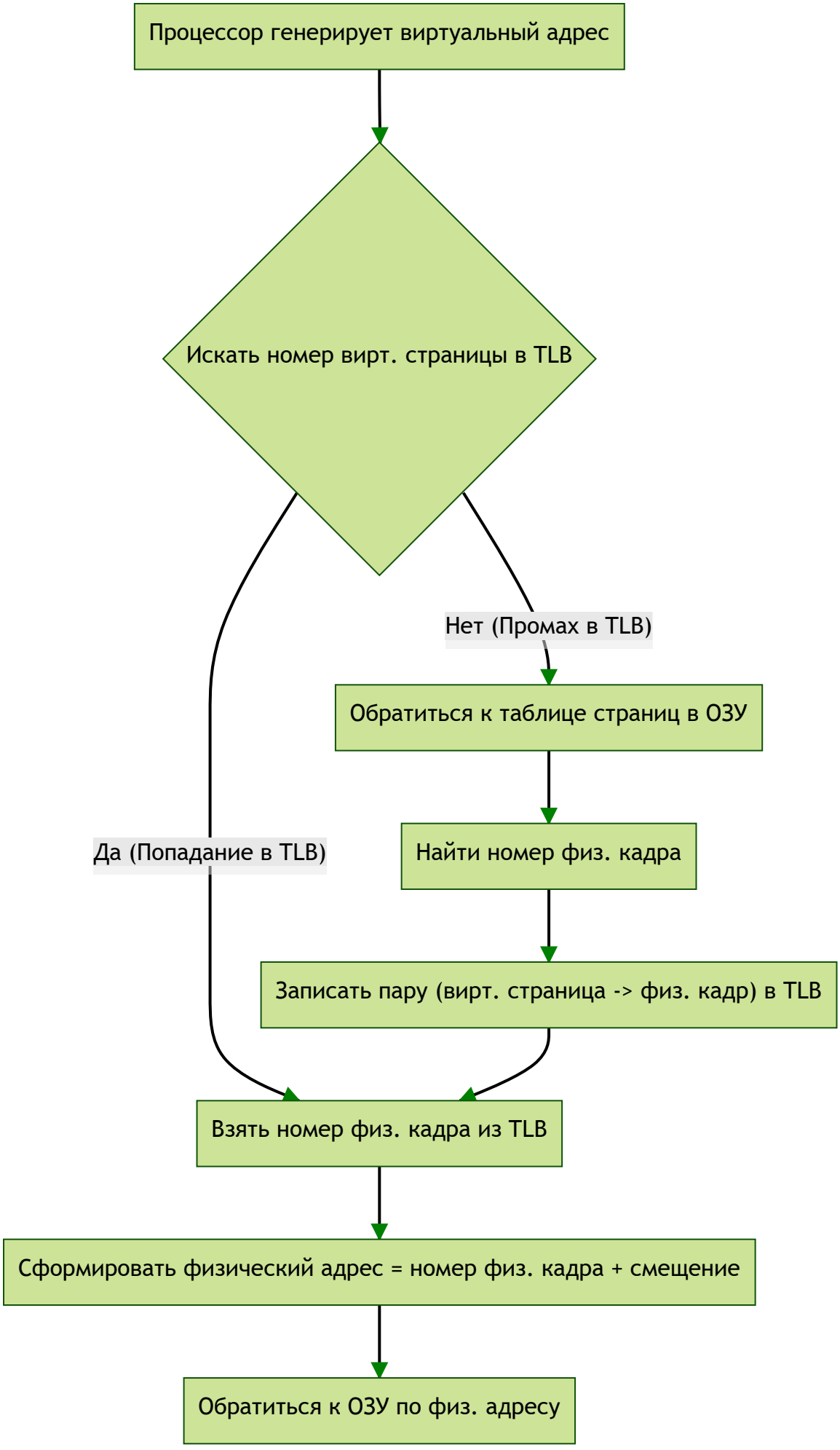
1. Процессор генерирует виртуальный адрес.
2. MMU извлекает из него номер виртуальной страницы.
3. MMU обращается к таблице страниц (адрес которой хранится в специальном регистре, например, `%cr3` в x86) и находит запись, соответствующую этому номеру.
4. В этой записи хранится номер физического кадра.
5. MMU конструирует физический адрес, объединяя номер физического кадра и исходное смещение внутри страницы.
6. Этот физический адрес выставляется на шину для обращения к ОЗУ.

Буфер трансляции (TLB - Translation Lookaside Buffer)

Обращение к таблицам страниц в ОЗУ для каждой трансляции адреса — это медленно. TLB — это аппаратный кэш для недавно использованных результатов трансляции.

- **Принцип:** Это небольшая, очень быстрая ассоциативная память. Перед обращением к таблицам страниц в ОЗУ, MMU сначала ищет нужный номер виртуальной страницы в TLB.
- **Попадание в TLB:** Если запись найдена, физический адрес получается мгновенно.
- **Промах в TLB:** Если запись не найдена, MMU выполняет медленный поиск по таблицам страниц в ОЗУ, а результат заносит в TLB для будущего использования.

Флоу-чарт процесса трансляции виртуального адреса в физический.


Процессор генерирует виртуальный адрес
Искать номер вирт. страницы в TLB
Нет (Промах в TLB)
Обратиться к таблице страниц в ОЗУ
Найти номер физ. кадра
Да (Попадание в TLB)
Записать пару (вирт. страница -> физ. кадр) в TLB
Взять номер физ. кадра из TLB
Сформировать физический адрес = номер физ. кадра + смещение
Обратиться к ОЗУ по физ. адресу

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

Понятие и классификация сетей ЭВМ

- **Сеть ЭВМ** — это система, состоящая из:
  - **Средств вычислительной техники (СВТ):** Компьютеры (узлы, хосты), которые обрабатывают информацию.
  - **Средств телекоммуникаций (СТК):** Каналы связи (кабели, радиоволны) и сетевое оборудование (коммутаторы, маршрутизаторы), которые обеспечивают передачу данных.
- **Классификация сетей:**
  - **По размеру:**
    - `PAN` (Personal Area Network): Персональная сеть (Bluetooth).
    - `LAN` (Local Area Network): Локальная вычислительная сеть (в пределах здания, кампуса).
    - `MAN` (Metropolitan Area Network): Городская сеть.
    - `WAN` (Wide Area Network): Глобальная сеть (Интернет).
  - **По принадлежности:** Офисные, корпоративные, частные.
  - **По топологии:** Шина, звезда, кольцо, смешанная.

Сообщение и пакет

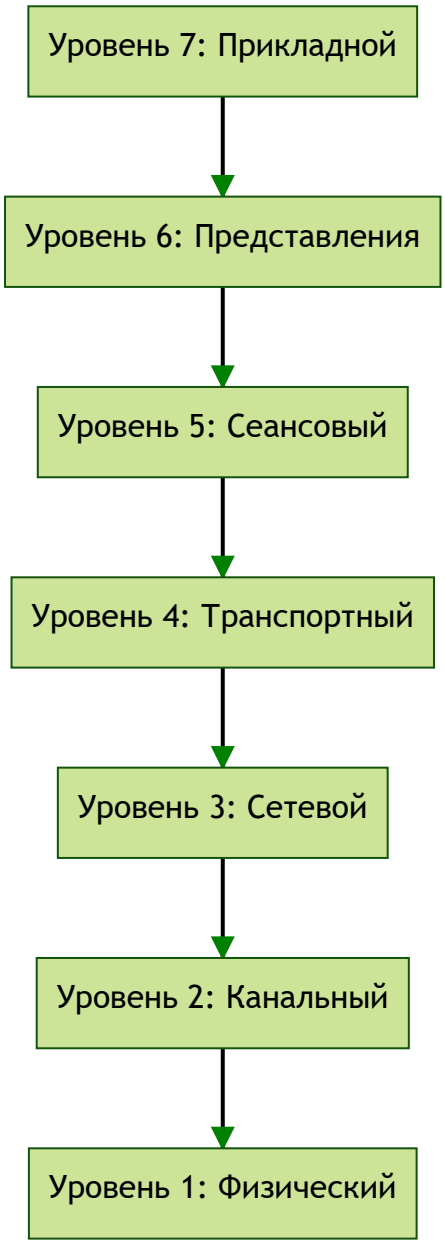
- **Сообщение:** Это полный блок данных, который приложение хочет передать (например, файл целиком или веб-страница).
- **Пакет:** Для передачи по сети большие сообщения разбиваются на небольшие блоки фиксированного или переменного размера, называемые пакетами (или кадрами, фреймами, сегментами в зависимости от уровня).
  - **Структура пакета:**
    - **Заголовок:** Содержит служебную информацию: адреса отправителя и получателя, номер пакета, тип протокола и т.д.
    - **Данные (Payload):** Непосредственно фрагмент исходного сообщения.
    - **Концевик (Trailer):** Содержит, как правило, контрольную сумму для проверки целостности пакета.
  - **Преимущества пакетной коммутации:** Эффективное использование каналов связи (канал не занимается одним сообщением надолго), повышение надежности (при ошибке нужно повторно передать только один маленький пакет, а не все сообщение).

Модель взаимодействия открытых систем (OSI)

OSI — это эталонная 7-уровневая модель, которая описывает, как данные должны передаваться между двумя системами в сети. Каждый уровень выполняет свою конкретную задачу и взаимодействует только с соседними уровнями (выше и ниже).

1. **Физический:** Передача сырых битов по физической среде (кабель, радио).
2. **Канальный:** Обеспечивает надежную передачу данных (кадров/фреймов) между двумя *соседними* узлами в одной локальной сети. Использует MAC-адреса.
3. **Сетевой:** Отвечает за маршрутизацию пакетов между различными сетями. Определяет путь от исходного узла до конечного. Использует IP-адреса.
4. **Транспортный:** Обеспечивает надежную доставку данных “от процесса к процессу”. Может гарантировать доставку и правильный порядок сегментов (TCP) или просто отправлять данные без гарантий (UDP).
5. **Сеансовый:** Управление сеансами связи между приложениями.
6. **Представления:** Преобразование данных в понятный для приложения формат (например, сжатие, шифрование, обработка кодировок).
7. **Прикладной:** Непосредственно протоколы, с которыми работают пользовательские приложения (HTTP, FTP, SMTP).

7-уровневая модель OSI.



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

Модель TCP/IP — это более практическая 4-уровневая модель, которая используется в реальном интернете.

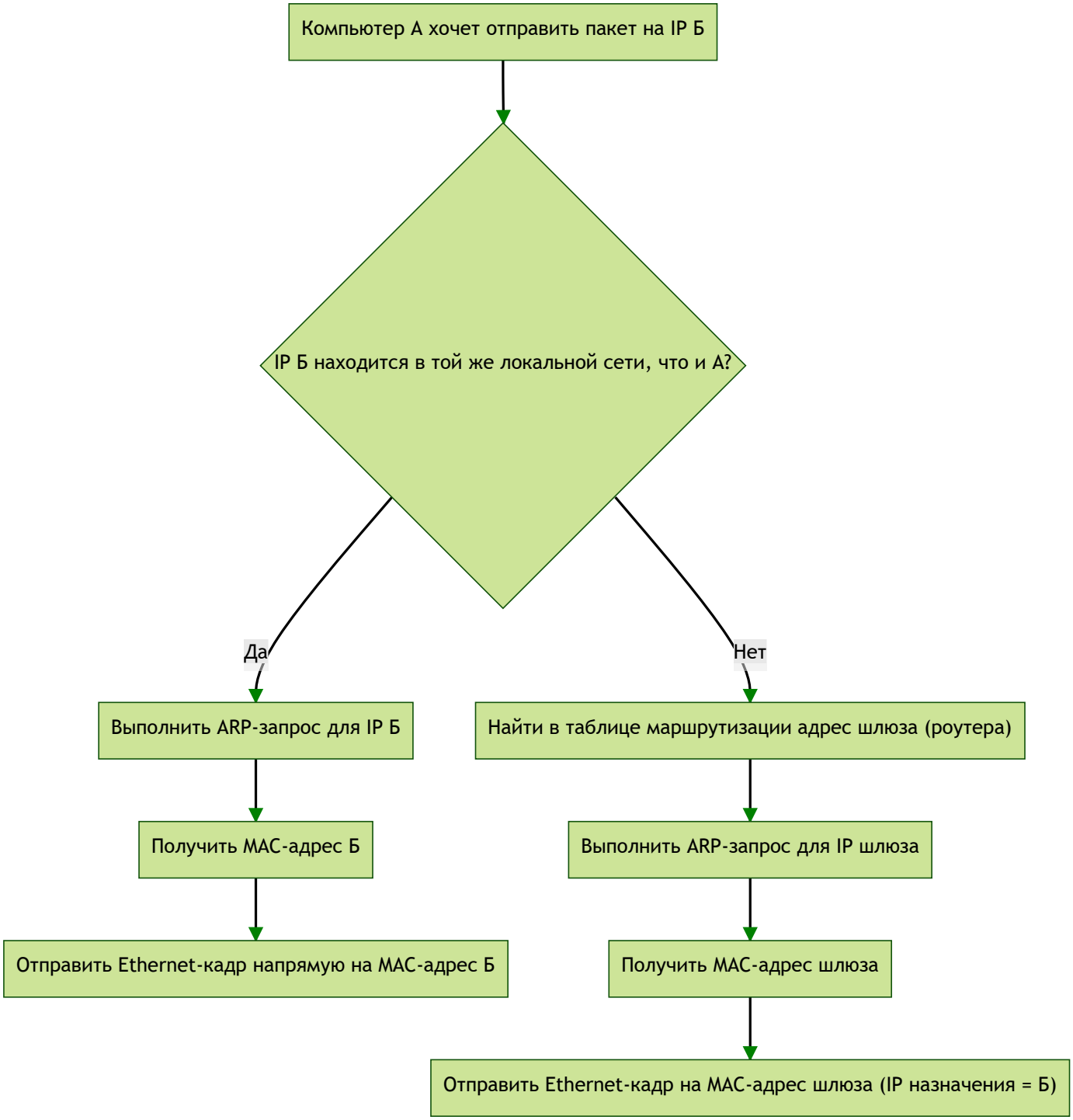
1. Канальный уровень (Link Layer)

- **Задачи:** Передача данных (кадров) в пределах одной физической сети (LAN).
- **Протоколы:** Ethernet, Wi-Fi.
- **Адресация:** Используются **MAC-адреса** — уникальные 48-битные идентификаторы, “зашитые” в каждую сетевую карту производителем.
- **ARP (Address Resolution Protocol):** Протокол, который позволяет узнать MAC-адрес узла, зная его IP-адрес. Узел посылает широковещательный ARP-запрос “Кто имеет IP-адрес X.X.X.X?”, и нужный узел отвечает своим MAC-адресом.

2. Сетевой уровень (Internet Layer)

- **Задачи:** Маршрутизация пакетов между различными сетями.
- **Протокол: IP (Internet Protocol)**.
- **Адресация:** Используются **IP-адреса** — логические 32-битные (IPv4) или 128-битные (IPv6) адреса, которые назначаются узлам администратором сети или по DHCP.
- **Передача и маршрутизация:**
  1. Компьютер-отправитель (A) хочет отправить пакет компьютеру-получателю (B).
  2. Он смотрит в свою **таблицу маршрутизации**.
  3. **Случай 1: B находится в той же локальной сети.** ОС определяет MAC-адрес B с помощью ARP и отправляет IP-пакет, инкапсулированный в Ethernet-кадр, напрямую на MAC-адрес B.
  4. **Случай 2: B находится в другой сети.** ОС видит, что адрес B не относится к его локальной сети. В таблице маршрутизации находится маршрут по умолчанию (**default gateway**) — это IP-адрес маршрутизатора (роутера). ОС определяет MAC-адрес роутера с помощью ARP и отправляет пакет на *MAC-адрес роутера*, но с *IP-адресом назначения B*.
  5. Роутер, получив пакет, смотрит на IP-адрес назначения, и на основе своей таблицы маршрутизации пересылает пакет следующему роутеру на пути к сети назначения. Этот процесс повторяется, пока пакет не достигнет роутера в сети B, который уже доставит его напрямую на B.

Флоу-чарт принятия решения о маршрутизации.



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

Выделение адресов (DHCP)

- **DHCP (Dynamic Host Configuration Protocol):** Протокол, который позволяет компьютерам в сети автоматически получать IP-адреса и другие сетевые настройки (маску подсети, адрес шлюза, адреса DNS-серверов) от специального DHCP-сервера. Это избавляет администратора от необходимости настраивать каждый компьютер вручную.

Сервисы имен (DNS)

- **Проблема:** Людям неудобно запоминать IP-адреса, удобнее использовать доменные имена (например, `se.ifmo.ru`).
- **DNS (Domain Name System):** Это иерархическая распределенная система, которая преобразует доменные имена в IP-адреса и обратно.
  - **Принцип работы:** Когда вы вводите в браузере `se.ifmo.ru`, ваш компьютер отправляет DNS-запрос на DNS-сервер. Сервер ищет в своей базе (или опрашивает другие серверы) IP-адрес, соответствующий этому имени, и возвращает его вашему компьютеру, после чего браузер может установить соединение.

Транспортный уровень

- **Задача:** Обеспечение связи между конкретными программами (процессами) на хостах отправителя и получателя. Использует **порты** для идентификации программ.
- **TCP (Transmission Control Protocol):**
  - **Надежный, с установлением соединения.** Перед передачей данных устанавливается “виртуальное соединение” (трехстороннее рукопожатие).
  - Гарантирует, что все данные будут доставлены в правильном порядке и без ошибок (за счет подтверждений и повторных передач).
  - Используется для приложений, где важна целостность данных: HTTP (веб), FTP (файлы), SMTP (почта).
- **UDP (User Datagram Protocol):**
  - **Ненадежный, без установления соединения.** Просто отправляет пакеты (датаграммы) без каких-либо гарантий. Пакеты могут теряться, дублироваться или приходить не по порядку.
  - Контроль надежности полностью ложится на плечи прикладной программы.
  - Используется там, где важна скорость, а не 100% надежность: DNS, VoIP (голосовая связь), онлайн-игры.

Прикладной уровень

Это уровень, на котором работают протоколы, непосредственно используемые приложениями. Программист, разрабатывая сетевое приложение, реализует один из существующих протоколов или создает свой собственный поверх TCP или UDP.

- **Примеры:** HTTP, FTP, SMTP, DNS, SSH.

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

Интерфейс и уровни стандартизации

Интерфейс ввода-вывода — это совокупность правил и средств, обеспечивающих обмен данными между процессором и внешним устройством.

- **Уровни стандартизации:**
  1. **Конструктивный:** Физические размеры и форма разъемов, количество контактов (например, USB Type-A, RJ-45).
  2. **Физический/Электрический:** Уровни напряжений, частота сигналов, тип среды передачи.
  3. **Логический:** Протокол обмена — последовательность сигналов, форматы команд и данных.

Сопряжение и циклы обмена

- **Сопряжение:** Контроллер ВУ подключается к **системной шине** компьютера. Это позволяет процессору обращаться к регистрам контроллера так же, как к ячейкам памяти (если I/O отображается на память) или через специальные команды `IN`/`OUT` (если пространство I/O отдельное, как в БЭВМ).
- **Циклы обмена:** Обмен данными между процессором и регистрами контроллера происходит по стандартным временным диаграммам чтения/записи на системной шине. Контроллер, в свою очередь, обменивается данными с внешним устройством по своему собственному **периферийному интерфейсу** (например, USB, SATA).

Регистры контроллера

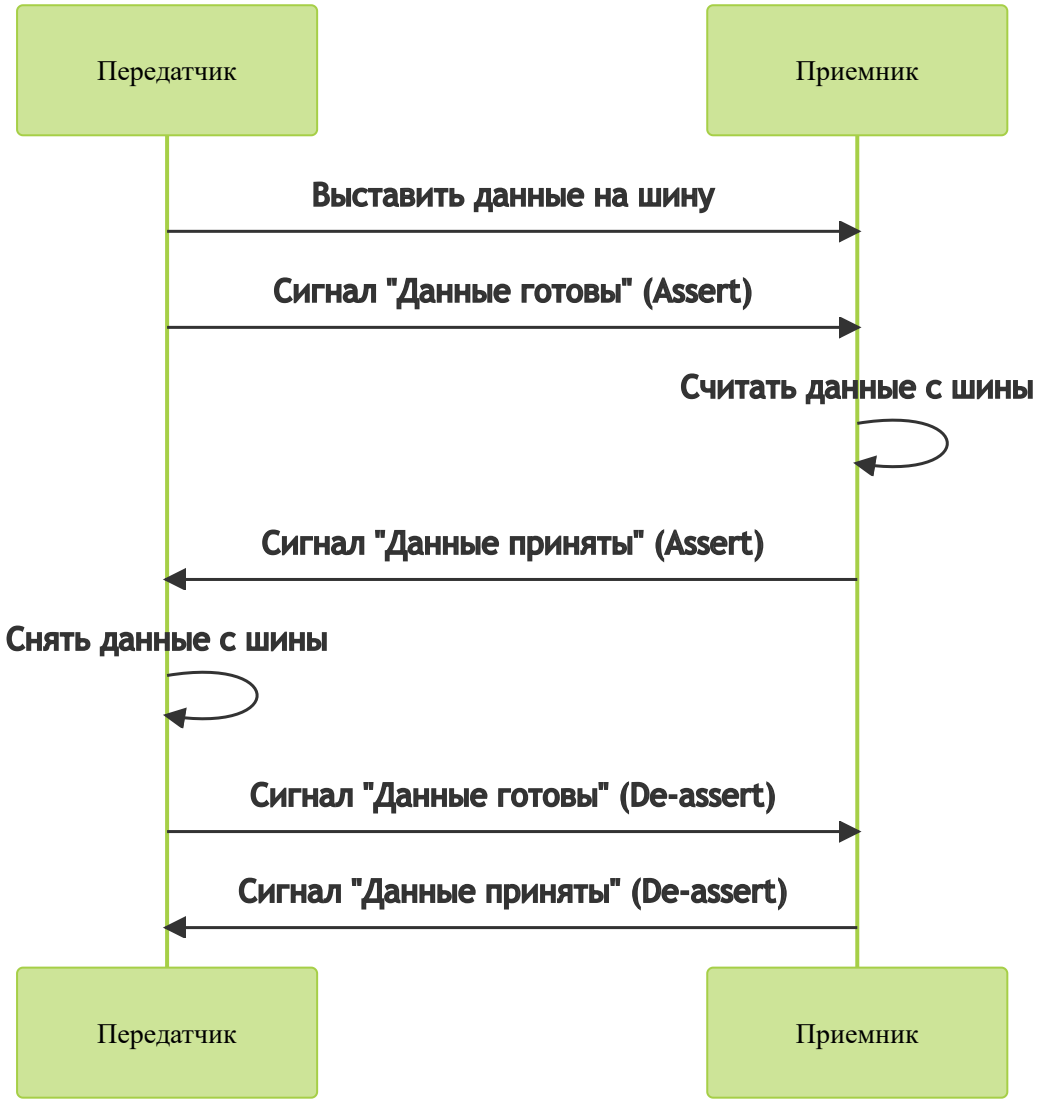
- **Регистр данных (РД):** Буфер для передачи данных. Процессор пишет в него данные для вывода или читает из него данные, полученные от устройства.
- **Регистр состояния/статуса (РС):** Хранит информацию о текущем состоянии устройства (готов, занят, ошибка). Процессор читает этот регистр, чтобы узнать, можно ли начать обмен.
- **Регистр управления:** Процессор пишет в этот регистр команды для устройства (например, начать печать, разрешить прерывания, установить скорость).

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

- **Принцип:** Все биты одного байта или слова передаются **одновременно** по отдельным параллельным проводам.
- **Преимущества:** Высокая скорость передачи данных.
- **Недостатки:** Требует много проводов, что делает кабели дорогими и громоздкими. На больших расстояниях и высоких частотах возникает проблема рассинхронизации сигналов на разных проводах.
- **Контроллер параллельной передачи/приема (асинхронный):**

- **Структура:** Основные элементы — регистр данных и управляющий RS-триггер.
- **Протокол (Handshake):**
  1. **Передатчик** записывает данные в свой регистр и выставляет на линию сигнал “Данные готовы”.
  2. **Приемник**, увидев этот сигнал, считывает данные со своих входов в свой регистр.
  3. **Приемник** выставляет на вторую линию ответный сигнал “Данные приняты”.
  4. **Передатчик**, получив подтверждение, снимает сигнал “Данные готовы” и может отправлять следующую порцию данных.
- Такой асинхронный обмен с квитированием (подтверждением) обеспечивает надежную передачу между устройствами с разной скоростью.

Диаграмма последовательности для протокола “рукопожатия”.



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

- **Принцип:** Биты данных передаются **последовательно**, один за другим, по одной линии.
- **Синхронность:** Используется дополнительная линия — **линия тактовых импульсов (синхроимпульсов)**. Передатчик генерирует тактовые импульсы, и приемник считывает очередной бит данных по каждому импульсу.
- **Преимущества:** Требуется мало проводов (данные + такт + земля).

- **Недостатки:** Скорость ниже, чем у параллельного интерфейса.
- **Контроллер синхронной последовательной передачи/приема:**
  - **Структура:** Ключевые элементы — **сдвиговый регистр** и **счетчик**.
  - **Передача:**
    1. Процессор записывает байт данных в буферный регистр контроллера.
    2. Данные из буфера загружаются в сдвиговый регистр.
    3. Контроллер начинает генерировать тактовые импульсы. С каждым импульсом содержимое сдвигового регистра сдвигается на один бит, и крайний бит выходит на линию данных.
    4. Счетчик считает импульсы. Когда будет передано 8 бит, контроллер останавливается или загружает следующую порцию данных.
  - **Прием:** Процесс обратный. С каждым тактовым импульсом от передатчика бит с линии данных “вдвигается” в сдвиговый регистр приемника. После 8 импульсов в регистре оказывается принятый байт.

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

- **Проблема:** Как избавиться от отдельной линии тактовых импульсов?
- **Решение:** Передатчик и приемник имеют свои собственные, независимые, но очень точные тактовые генераторы, работающие на **одинаковой, заранее согласованной частоте** (скорости, например, 9600 бод).
- **Проблема синхронизации:** Даже при одинаковой частоте генераторы имеют небольшой фазовый сдвиг, который со временем накапливается и приводит к ошибкам.
- **Решение:** Данные передаются не сплошным потоком, а короткими **кадрами (фреймами)**. Каждый кадр синхронизируется отдельно.
- **Формат кадра:**
  - **Стартовый бит (Start bit):** Всегда равен 0. Сигнализирует приемнику о начале передачи кадра. Приемник, увидев спад на линии, запускает свой внутренний механизм приема.
  - **Биты данных:** 7 или 8 бит полезной информации.
  - **Бит четности (Parity bit, опционально):** Используется для простого контроля ошибок.
  - **Стоповый бит (Stop bit):** Всегда равен 1. Гарантирует, что линия вернется в состояние покоя (1) перед началом следующего кадра.
- **Принцип деления частоты:** Для надежного приема приемник использует тактовый генератор с частотой, кратной скорости передачи (например, в 16 раз выше). Получив стартовый бит, он отсчитывает 8 тактов (до середины бита) и считывает его значение, затем отсчитывает еще 16 тактов и считывает середину следующего бита и т.д. Это позволяет нивелировать небольшой фазовый сдвиг.

38. Контроллер передачи асинхронного последовательного интерфейса.

39. Контроллер приема асинхронного последовательного интерфейса.

Эти два вопроса описывают реализацию UART (Universal Asynchronous Receiver-Transmitter).

- **Контроллер передачи:**

1. Получает байт данных от процессора.
2. Помещает его в сдвиговый регистр.
3. Добавляет к данным стартовый и стоповый биты.
4. Начинает побитово “выталкивать” сформированный кадр на линию передачи с заданной скоростью.

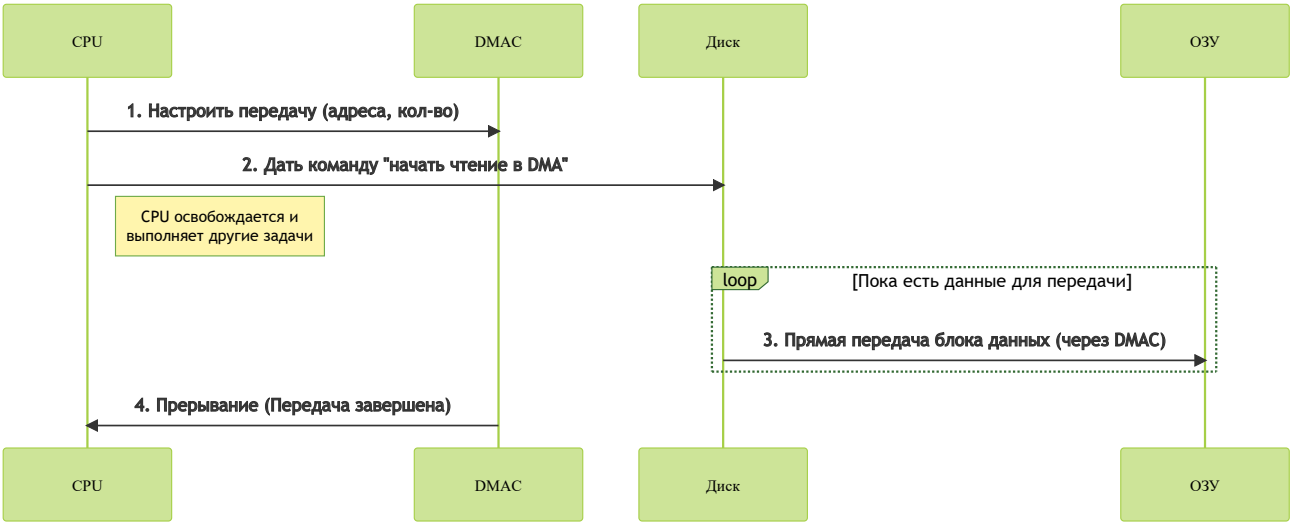
- **Контроллер приема:**

1. Постоянно следит за линией данных.
2. Обнаружив спад (стартовый бит), запускает механизм приема.
3. Используя свой генератор и делитель частоты, считывает биты данных в середине каждого битового интервала и задвигает их в сдвиговый регистр.
4. Проверяет стоповый бит. 5. Когда весь байт принят, выставляет флаг готовности для процессора.

40. Организация прямого доступа к памяти. Контроллер ПДП.

- **Проблема:** Программно-управляемый ввод-вывод, даже с прерываниями, требует участия процессора для пересылки каждого байта/слова между контроллером и памятью. Для высокоскоростных устройств (например, дисков) это создает большую нагрузку на CPU.
- **ПДП (Прямой доступ к памяти, DMA - Direct Memory Access):** Это механизм, который позволяет контроллеру ВУ обмениваться данными с оперативной памятью **напрямую, без участия процессора**.
- **Контроллер ПДП (DMAC):** Специализированный процессор, который управляет этим процессом.
  - **Принцип работы:**
    1. CPU **программирует** контроллер ПДП: сообщает ему, какую операцию выполнить (чтение с диска в память или наоборот), начальный адрес в памяти, начальный адрес на диске и количество байт для передачи.
    2. После этого CPU **освобождается** и может выполнять другие задачи.
    3. Контроллер ПДП берет управление шиной на себя (захватывает шину) и организует пересылку данных напрямую между контроллером диска и ОЗУ.
    4. Когда вся передача завершена, контроллер ПДП посылает процессору **прерывание**, сообщая об окончании операции.
- **Преимущества:** Значительно разгружает центральный процессор, позволяя одновременно выполнять вычисления и высокоскоростной ввод-вывод. Это критически важно для производительности современных систем.

Диаграмма последовательности для операции DMA.



Источник — https://xn--b1amah.xn--80aalyho.xn--d1acj3b/mediawiki/index.php?title=ОПД:Экзамен_Сем2&oldid=171