Оглавление

[Глава 1. Вычислители 5](#_30j0zll)

[1.1. Электронная вычислительная машина. Классификация ЭВМ 5](#_1fob9te)

[1.2. Архитектура вычислителя и его расширяющие компоненты 10](#_tyjcwt)

[Особенности ОЗУ: 23](#_4d34og8)

[Типы оперативной памяти: 23](#_2s8eyo1)

[1.3. Специализированные вычислители 30](#_17dp8vu)

[1.4 Вычислители с множественными потоками команд и одиночным потоком данных (MISD), истинно параллельные процессоры (MIMD), систолические вычислители 38](#_26in1rg)

[1.5. Разрядность элементов вычислителя 41](#_1ksv4uv)

[1.6. Программная среда специализированного вычислителя 49](#_4i7ojhp)

[Глава 2. Процессор (компьютер) 52](#_2xcytpi)

[2.1. Узлы процессора 52](#_1ci93xb)

[2.2. Ядро процессора 64](#_qsh70q)

[2.3. Представление о микросхемах обвязки процессоров 73](#_23ckvvd)

[2.4. Представление о конвейерном выполнении команд в процессорах 77](#_32hioqz)

[2.6. Классификация и особенности шин в отношении способа передачи, метода синхронизации, топологии, способа управления, адресации устройств 86](#_4f1mdlm)

[2.7. Виртуальные машины. Виртуализация и эмуляция. Гипервизор и супервизор. Контейнеры и докеры. 96](#_2u6wntf)

[2.8. Ядро ОС. Планировщик и механизмы переключения задач 104](#_nmf14n)

[Глава 3. Процессор (программирование) 109](#_37m2jsg)

[3.1. Понятие о параллельных и распределенных вычислениях. Программно-аппаратные переходы и аспекты их применения. 109](#_1mrcu09)

[3.2.Компиляция и интерпретация исходных текстов программ. Объектный код, исполняемый код, динамическое и статическое связывание модулей программы. Байт-код. 118](#_4k668n3)

[3.3 Специализированные средства разработки программного обеспечения 120](#_2zbgiuw)

[3.9. Загрузчики и компоновщики. Форматы объектных модулей. Оверлейные загрузчики и оверлеи. 127](#_1egqt2p)

**3.2. Способы передачи параметров в функции и получения данных из функций. Единственность и множественность возвращаемых параметров функции. Описатель restrict.**

Глава 1. Вычислители

1.1. Электронная вычислительная машина. Классификация ЭВМ

*Определение 1.1.1*

**Электронная вычислительная машина (ЭВМ) или компьютер** — это автоматическое устройство, выполняющее обработку информации.

Компьютеры классифицируется по следующим критериям.

**Классификация компьютеров в отношении круга решаемых задач**:

Критерием деления вычислительных машин на эти три класса является форма представления информации, с которой они работают.

* Цифровые вычислительные машины (ЦВМ) – машины дискретного действия, то есть осуществляющие пошаговую работу с информацией, представленной в цифровой форме.
* Аналоговые вычислительные машины (АВМ) – вычислительные машины непрерывного действия, работают с информацией, представленной в аналоговой (непрерывной) форме, т.е. в виде непрерывного ряда значений какой-либо физической величины (чаще всего электрического напряжения).
* Гибридные вычислительные машины (ГВМ) – вычислительные машины комбинированного действия, работают с информацией, представленной и в цифровой, и в аналоговой форме; они совмещают в себе достоинства АВМ и ЦВМ. ГВМ целесообразно использовать для решения задач управления сложными быстродействующими техническими комплексами.

**Классификация компьютеров в отношение условий эксплуатации:**

* универсальные – рассчитаны для решения широкого круга задач при нормальных условиях эксплуатации.
* специальные(можно подразделить на мобильные, военные и т.д.) – предназначены для решения узкого круга задач или даже одной задачи, требующей неоднократного решения, и функционируют в специальных условиях использования, управляют технологическими установками. Применяются в операционных, на ракетах, самолетах и вертолетах, вблизи высоковольтных линий передач или в зоне действия радаров, радиопередатчиков, под водой на глубине, в условиях пыли, грязи, вибраций, взрывоопасных газов и т. п.

Машинные возможности специальных компьютеров, как правило, ограничены. Но благодаря их узкой ориентации заданный круг задач осуществляется наиболее эффективно.

**Классификация компьютеров в отношение архитектуры:**

*Определение 1.1.2*

**Архитектура компьютера** — логическая организация и структура аппаратных ресурсов вычислительной системы. Архитектура заключает в себе требования к функциональности и принципы организации основных узлов компьютера.

*Определение 1.1.3*

**Арифметико-логическое устройство** (АЛУ) — блок процессора, который под управлением устройства управления (УУ) служит для выполнения арифметических и логических преобразований (начиная от элементарных) над данными, называемыми в этом случае операндами.АЛУ занимается логическими и побитовыми операциями, а также арифметическими операциями над стандартными типами, кроме указательных (иногда и над ними)

В настоящее время наибольшее распространение в компьютерах получили два типа архитектуры:

* принстонская (фон Неймана)
* [гарвардская](http://ru.wikipedia.org/wiki/%D0%93%D0%B0%D1%80%D0%B2%D0%B0%D1%80%D0%B4%D1%81%D0%BA%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0).

Для описания каждого из типов введем еще несколько определений.

*Определение 1.1.4*

**Устройства ввода-вывода** (УВВ) — устройства, с помощью которых осуществляется взаимодействие компьютера и пользователя. Например, ввод информации в память компьютера производится с помощью мыши и клавиатуры, а вывод отображается на мониторе или печатается на принтере.  
С помощью **интерфейса ввода-вывода** (ИВВ) осуществляется обмен информацией между этими устройствами.

*Определение 1.1.5*

**Шина** — канал связи, который соединяет между собой все электронные компоненты компьютера.

На рисунках ниже присутствуют три типа шин. При помощи адресной шины формируется указатель на требуемые данные, а шина данных передает их компонентам компьютера. По шине управления передаются сиг­налы, которые определяют характер обмена информацией, например, сиг­налы чтения, записи, готовности.

*Определение 1.1.6*

**Устройство управления** (УУ) — узел процессора, который осуществляет остальными компонентами. Его задачи — выборка и декодирование потока инструкций, выдача кодов функций в исполнительные устройства, принятие решений по признакам результатов вычислений, синхронизация узлов процессора.

*\*что такое регистр см. опр 2.1.1*

Обе выделяют два основных узла компьютера: центральный процессор и память компьютера.

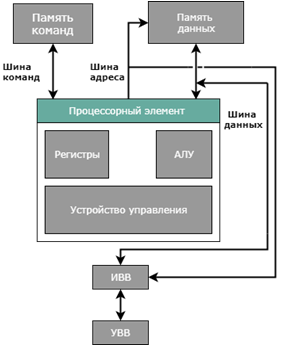
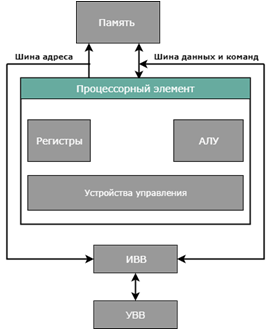


Рис.1.1.1 Рис.1.1.2

Принстонская архитектура (фон Неймана) (Рис.1.1.1) 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 команды, и данные. Классическая гарвардская архитектура (Рис.1.1.2) отличается использованием разделенной памяти.

Различие — в структуре памяти: в принстонской (фон Неймана) архитектуре программы и данные хранятся в одном массиве памяти и передаются в процессор по одной шине, тогда как гарвардская архитектура предусматривает отдельные хранилища и потоки передачи для команд и данных.

**Классификации компьютеров в отношение производительности и области применения:**

По производительности и характеру использования компьютеры можно условно подразделить на:

*Определение 1.1.4*

**Микрокомпьютер** — компьютер, полностью расположенный на одной микросхеме.

Продвинутые модели микрокомпьютеров имеют несколько процессоров. Производительность компьютера определяется еще и емкостью оперативной памяти, типами периферийных устройств, качеством конструктивных решений и др.

*Определение 1.1.5*

**Микроконтроллер** — специализированный компьютер или его часть, которая обеспечивает работоспособность какого-либо узла или устройства компьютера.

*Определение 1.1.6*

**Персональные компьютеры** — компьютеры, рассчитанные на использование малым ограниченным количеством людей.

*Определение 1.1.7*

**Рабочие станции** — персональные мощные компьютеры, специализированных на выполнении круга рабочих задач.

*Определение 1.1.8*

**Серверы** – многопользовательские мощные компьютеры в вычислительных сетях, выделенные для обработки запросов от всех станций сети.

Можно выделить следующие группы в классе компьютеры:

* мэйнфреймы — распределенный компьютер, в том смысле, что разные узлы одного компьютера, в силу своих размеров, оформлены в виде отдельных блоков. Применение мэйнфреймов связано с интенсивными вычислениями и обработкой больших объемов информации.
* миникомпьютеры — компьютер небольших размеров, обладающий более низкими параметрами, чем мэйнфрейм, к которому могут подключаться десятки и сотни устройств или микрокомпьютеров.
* суперкомпьютеры — большое число серверов, соединенных локальной высокоскоростной шиной для достижения максимальной производительности. Используются для работы с приложениями, требующими наиболее интенсивных вычислений (например, прогнозирование погодно-климатических условий, моделирование ядерных испытаний и т.п.).   
    
  **Критерии выбора решений:**

(Полностью зависит от поставленной задачи)

Основные параметры для выбора решений:

* производительность
* стоимость
* надежность
* масса
* габариты

1.2. Архитектура вычислителя и его расширяющие компоненты

*Определение 1.2.1*

Под **архитектурой вычислительного средства** – понимается совокупность общих принципов организации аппаратно-программных средств и их характеристик, определяющая функциональные возможности компьютера при решении соответствующих классов задач.

**Обязательные компоненты вычислителя:**

* процессоры
* энергонезависимая память
* системы управления и ввода-вывода информации

*Определение 1.2.2*

**Процессор** – функциональная часть компьютера, выполняющая основные операции по обработке данных и управлению работой других блоков.

Процессор является преобразователем информации, поступающей из памяти и внешних устройств. Наиболее важными частями процессора являются арифметико-логическое устройство (АЛУ) и устройство управления (УУ).

***1.2.1. КЭШ***

*Определение 1.2.5*

**Кэш-память** (англ. *cache* – тайник) — это высокоскоростная память произвольного доступа, не имеющая адресов и используемая процессором компьютера для временного хранения информации.

Кэш-памяти служит своеобразным буфером между различными устройствами для временного хранения и обработки информации. Время доступа к информации, хранящейся в кэш-памяти, меньше, чем время доступа к этой же информации, хранящейся в других видах памяти компьютера.

В зависимости от назначения кэш-память представляется в следующей **иерархии**:

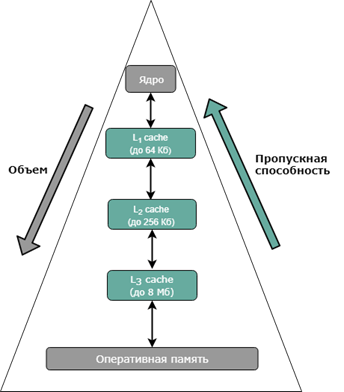


Рис. 1.2.1. Уровни кэш-памяти

1. Кэш первого уровня (L1) — наиболее быстрый и наименьший по объему, имеющий время доступа порядка нескольких тактов и размер в десятки Кбайт. Содержит данные, чаще всего используемые процессором, работает на частотах близких ему и напрямую с его ядрами, причем каждое из ядер имеет доступ только к своей микросхеме L1. Также является буфером между процессором и кэш-памятью второго уровня.
2. Кэш второго уровня (L2) — от 2 до 10 раз медленнее L1, но имеет более существенный объем (несколько сотен Кбайт). Предназначена для временного хранения важной информации, у которой вероятность запроса ниже, чем у информации из L1. Служит буфером между уровнями L1 и L3.
3. Кэш третьего уровня (L3)  — в массовых процессорах используется недавно. Он более медленный (время доступа около сотни тактов), чем два предыдущих, но гораздо быстрее оперативной памяти, объемом до нескольких десятков Мбайт. Данный уровень является общим для всех ядер процессора, в отличие от L2 и L3.   
   L3 обеспечивает взаимодействие ядер процессора между собой и хранит важные данные с относительно низкой вероятностью запроса.  
   При отсутствии данного уровня кэш-памяти L2 совмещает в себе функции L2 и L3.

**Тэги доступа**

Каждый структурный элемент кэш-памяти кроме байт данных оперативной памяти должен сохранять и его четырехбайтный адрес в оперативной памяти. Такое подобие строки называется кэш-строкой (cache-line).

В кэше каждой из кэш-строк дополнительно ставится в соответствие **тег** (tag), который является адресом продублированных в этой кэш-строке данных в основной памяти. Тег, кроме 24 бит, содержит бит значимости, определяющий действительность строки (если бит значимости 0, данная строка считается недействительной и обращения к данным не кэшируются).

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. Если есть совпадение с тегом какой-либо кэш-строки — это называется *попаданием в кэш* (*cache hit*), тогда производится чтение или запись данных в этой кэш-строке, иная ситуация называется *кэш-промахом* (*cache miss*).

Отношение количества попаданий в кэш к общему количеству запросов к памяти называют рейтингом попаданий (*hit rate*), оно является мерой эффективности кэша для выбранного алгоритма или программы.

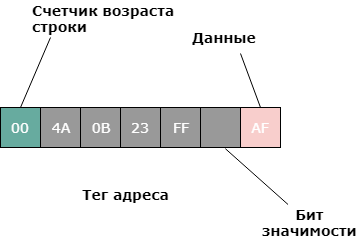


Рис. 1.2.2. Структура гипотетического кэша

**Загрузка и выгрузка**

Устройство, которое управляет содержанием кэша, получает необходимую информацию из оперативной памяти, передает ее процессору, и возвращает результаты вычислений в оперативную память, называется контроллером кэш памяти.

Если при обращении ядра к контроллеру за получением данных, их не оказывается в кэш-памяти, ядро ожидает поступления нужных данных из медленной оперативной памяти. Такая ситуация называется *кэш-промахом*.

Таким образом, основная задача кэш-контроллера – заполнить кэш актуальными данными, причем с минимальным количеством кэш-промахов.

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

**Варианты стратегий помещения данных в кэш-память**:

* кэширование по требованию (on demand): обращение к оперативной памяти (с последующим помещением копии данных в кэш) происходит при возникновении кэш-промаха;
* на основе прогноза о данных, которые потребуются в ближайшем будущем, кэш-контроллер заранее осуществляет их загрузку в кэш-память;
* упреждающая спекулятивная загрузка данных:
  + загрузка из оперативной памяти в кэш не только затребованных данных, а также соседних данных в порядке возрастания адресов (эффективно для последовательной обработки данных);
  + адрес следующей запрашиваемой ячейки памяти предсказывается на основе анализа предыдущих обращений (интеллектуальные стратегии Look Through и Look Aside);

Первая стратегия имеет достаточно высокую частоту кэш-промахов, вследствие чего не используется. Последняя стратегия имеет высокую эффективность и сводит частоту возникновения кэш-промахов к ничтожно малому значению.

Существует несколько вариантов **политики замещения**:

* замещаемые данные выбираются произвольным образом (Random);
* замещаются данные с самая низкой частота обращений (Least Frequently Used, LFU);
* замещаются данные, к которым дольше всего не обращались (Least Recently Used, LRU);
* замещаются данные, загруженные раньше всех (First Input First Output, FIFO или Least Recently Replaced, LRR);

Первый вариант самый простой и не интеллектуальный, при этом не применяется т.к. неэффективен. Алгоритмы же LRU и FIFO используются во всех современных процессорах.

**Поддержание когерентности кэш-памяти**

**Когерентность кэш-памяти** — алгоритм, который обеспечивает согласованность содержимого оперативной памяти и кэш-памяти, выполняемый кэш-контроллером.

Если бы оперативная память была доступна только на чтение, то копия данных в кэш-памяти была бы идентична оригиналу данных в оперативной памяти. Но поскольку оперативная память доступна не только для чтения, но и для записи данных, своевременная модификация данных в оперативной памяти *очень важна для корректной работы с данными.*

Действительно, к оперативной памяти могут обращаться не только процессор, но и периферийные устройства, а в многопроцессорных и многоядерных процессорах разные процессоры или ядра. Допустим, к некоторой ячейке памяти, уже модифицированной в кэше, но еще не выгруженной в основную память, обращается периферийное устройство или другой процессор. В этом случае кэш-контроллер должен сначала обновить содержимое соответствующей ячейки оперативной памяти, иначе оттуда будут считаны неактуальные данные. Аналогично, если периферийное устройство или другой процессор изменяет содержимое ячейки оперативной памяти, кэш-контроллер должен выяснить, загружены ли модифицированные ячейки в его кэш-память, и если да, то их необходимо обновить.

Значит нужно вовремя отслеживать изменение данных в кэше и оперативной памяти и синхронизировать эти изменения — поддерживать когерентность между кэшем и оперативной памятью.

**Пути реализации когерентности:**

1) **Сквозная запись (Write True)** —самое простое, но самое неэффективное решение поддержки когерентности. Оперативная память кэшируется только на чтение, а запись проводится напрямую, минуя кэш, сразу в оперативную память. В силу своей неэффективности этот алгоритм не используется в современных процессорах.

2) **Алгоритм обратной записи (Write Back)** — более сложный. Для отслеживания операций модификации данных в кэш-памяти каждая кэш-строка имеет специальный флаг состояния. Если данные кэш-строки подверглись модификации, то флаг устанавливается в состояние «модифицированное» (Dirty). При обращении к памяти периферийного устройства кэш-контроллер проверяет, имеются ли запрашиваемые данные в кэше, и если да, то по флагу модификации определяется, актуальны ли данные в оперативной памяти (были модифицированы или нет). Если значение флага “Dirty ”, то в оперативной памяти данные не актуальны и содержимое соответствующей кэш-строки выгружается в оперативную память, а флаг устанавливается в состояние «не модифицировано» (Clear). При замещении кэш-строк кэш-контроллер также проверяет сначала состояние флага, и если строка была модифицирована, то она прежде выгружается в оперативную память.

3) **Протокол MESI** используется для реализация когерентности в современных процессорах. Он является вариантом алгоритма Write Back, но с четырьмя возможными значениями флага модификации кэш-строки:

* Модифицированная (M, Modified). Строка только в одном кэше может быть отмечена этим флагом. Что означает, что данная строка была изменена, но до памяти эти изменения еще не дошли. Хозяин такой строки без опроса остальных может спокойно читать и писать в нее.
* Эксклюзивная (E, Exclusive). Данные этой строки эксклюзивны и не присутствуют в остальных кэшах, кроме того они полностью совпадают с данными в оперативной памяти. Записывать и читать из нее можно без внешних запросов, так как она хранится только в одном кэше. После записи такая строка должна быть помечена, как модифицированная.
* Разделяемая (S, Shared). Строка может одновременно содержаться в кэшах нескольких устройств и использоваться совместно. Запросы на запись в такую строку направляются на общую шину, вследствие чего все строки с таким адресом помечаются как недействительные в остальных кэшах. При этом содержание основной памяти также обновляется. Чтение же из такой строки, не требует никаких внешних запросов.
* Недействительная (I, Invalid). Строка является недопустимой и попытка прочитать ее содержимое приведет к кэш-промаху. Она помечается недействительной, в случае если пуста или содержит неактуальную информацию.

**1.2.2. ПДП**

**Прямой доступ к памяти- это главный способ разгрузить процессор от операций обмена с памятью.**

Определение процессора см. в главе 1.2

*Определение 1.2.3*

**Сопроцессор** — это специализированный процессор, расширяющий возможности центрального процессора компьютерной системы, но оформленный как отдельный функциональный модуль. Физически сопроцессор может быть отдельной микросхемой или может быть встроен в центральный процессор.

Информация, хранимая во внешних устройствах памяти большой емкости, таких, как накопители на магнитных дисках и лентах, организована в виде блоков размером единицы и более килобайт. Для обмена данными между указанными устройствами памяти и основной (оперативной) памятью микропроцессора не подходят ни программный способ обмена, ни прерывания. Это обусловлено тем, что обмен производится блоками фиксированного размера в строгой последовательности, соответствующей расположению информации на магнитном носителе. Время на обмен одного байта данных строго фиксировано, ограничено скоростью передвижения носителя относительно магнитных головок и составляет весьма малое значение (единицы микросекунд и меньше). При программно-управляемом обмене и обмене с использованием прерываний на передачу байта данных затрачивается большее время. Для обмена данными в указанных условиях организуется прямой доступ к памяти (ПДП), или Direct Memory Access (DMA).

*Определение 2.3.8*

**Контроллер** (англ. controller – регулятор) — это электронное устройство, предназначенное для подключения к магистрали компьютера разных по принципу действия, интерфейсу и конструктивному исполнению периферийных устройств. Т.е. это микросхема, которая управляет каким-либо подключенным к процессору устройством (например, жестким диском). В задачи контроллера входит получение от программного обеспечения различных команд и передача их устройству.

*Определение 1.2.4.*

**Канал прямого доступа к памяти** — средства, позволяющие осуществить быстрый обмен данными непосредственно между основной памятью и внешним устройством (ВУ) без участия процессора. При этом способе обмена процедура ввода/вывода полностью осуществляется аппаратными средствами и возлагается на контроллер ПДП.

Обычный обмен между внешним устройством и памятью реализуется за два командных цикла: вначале данные поступают от источника в центральный процессор, а затем — из [процессора](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html) в приемник. При ПДП данные не проходят через [процессор](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html), и передача слова производится за один цикл. Поэтому основное достоинство обмена по каналу ПДП — высокая скорость обмена, ограниченная только временем доступа к памяти.

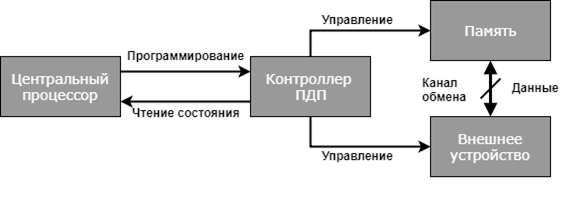


Рис. 1.2.2.1. Общий принцип организации канала прямого доступа к памяти

**Принципы организации ПДП.**

Структурная схема организации канала ПДП показана на рис. 3.10.1. Прямой доступ к памяти предоставляется по завершении текущего машинного цикла [процессора](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html). В отличие от [прерывания](http://pue8.ru/protsessory/681-sistemy-preryvanij.html) обмен по каналу ПДП выполняется без участия программы, поэтому содержимое рабочих регистров процессора сохраняется и на вхождение в режим ПДП не требуется затрат времени (отсутствует необходимость хранения в стеке содержимого рабочих регистров процессора).

Центральный [процессор](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html) выполняет программирование контроллера, настраивая его на требуемый режим работы, и следит за состоянием контроллера. Во время обмена данными по каналу ПДП процессор отключен, а контроллер вырабатывает сигналы управления обменом для памяти и внешнего устройства. Связь внешнего устройства памятью осуществляется по шинам адреса и данных системного интерфейса.

Проблема совместного использования шин центрального процессора и внешнего устройства решается двумя способами: организацией режима обмена с «захватом цикла» и (пакетного, или непрерывного) режима с блокировкой центрального процессора.

В режиме с «захватом цикла» обмен ведется одиночными передачами, когда для прямого доступа к памяти выделяются отдельные циклы (такты), т. е. передача данных (слов) перемежается с выполнением программы.

Один из вариантов обмена с «захватом цикла» состоит в использовании тех тактов, в которых центральный процессор не обменивается данными с памятью. Такие такты должны быть известны контроллеру ПДП. Некоторые [процессоры](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html) вырабатывают специальный сигнал, указывающий используется ли [процессором](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html) в данном цикле память. Применение рассмотренного способа организации обмена не снижает производительности [процессоров](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html), однако:

● требует дополнительных аппаратных затрат и позволяет реализовать только случайные, нерегулярные передачи;

● скорость обмена будет не быстрой, темп обмена нерегулярен, так как длительности циклов различных команд различны, и, кроме того, прямой доступ может все–таки замедлить выполнение программы, если цикл ПДП не превышает интервал, соответствующий такту процессора.

Более распространенным является **вариант способа с «захватом цикла»**, при котором центральный процессор принудительно отключается от системных шин адреса и данных. Его реализация связана с введением двух линий для передачи сигналов запроса на захват шин (ЗЗхв) и подтверждения захвата (ПЗхв). Сигнал ЗЗхв формируется контроллером ПДП.

После получения сигнала ЗЗхв процессор:

● приостанавливает выполнение очередной команды, не дожидаясь ее завершения;

● выдает в системный интерфейс сигнал подтверждения захвата ПЗхв;

● отключается от шин адреса и данных, переводя в высокоомное состояние шинные формирователи.

После получения сигнала ПЗхв контроллер ПДП использует шины системного интерфейса для обмена байтом или словом между ВУ и памятью. Затем снимает сигнал запроса ЗЗхв и возвращает управление шинами центральному процессору. Подготовив очередной байт или слово данных, контроллер ПДП вновь посылает сигнал ЗЗхв [процессору](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html) и т. д.

Как уже отмечалось, режим ПДП не требует сохранения состояния регистров процессора в стеке. Поэтому передача данных с «захватом цикла» происходит с большей скоростью, чем при обмене в режиме прерываний.

Обмен данными по каналу ПДП требует предварительной подготовки контроллера. Она заключается в том, что программа загрузки устанавливает необходимые параметры для передачи:

● количество байтов (слов) данных, которые должны быть переданы;

● начальный адрес передаваемых данных (адрес первого байта или слова);

● направление передачи (запись/чтение).

Для занесения этих параметров в контроллере предусмотрены регистр адреса и счетчик байтов (слов).

**1.2.3. Энергонезависимая память**

*Определение 1.2.5*

**Энергонезависимая память** — это компьютерная память, которая может хранить информацию при отсутствии питания (флеш-память, жесткий диск и т.д.).

*Определение 1.2.6*

**Постоянное запоминающее устройство (ПЗУ)** — энергонезависимая память, используется для хранения массива неизменяемых данных.

Она включает в себя все виды памяти на ПЗУ (ROM, Read–Only Memory) и ППЗУ (EPROM, Erasable Programmable Read–Only Memory). Это является главным преимуществом энергонезависимой памяти над оперативной, которая носит временный характер, т.е. при выключении компьютера все данные, с которыми работал пользователь, исчезают. Кроме того, максимальная емкость оперативной памяти составляет всего несколько Гбайт, тогда как объем энергозависимой памяти может достигать несколько Тбайт (пример жесткий диск), но при этом, как правило, значительно уступает оперативной памяти в быстродействии.

*Определение 1.2.7.*

Флэш-память (англ. *flash memory*) — тип долговременной памяти, информацию которой можно удалить или перепрограммировать электрическим методом.

Использование флэш-памяти в качестве мобильного носителя информации или в качестве хранилища программного обеспечения цифровых устройств это два определяющих способа применения флэш-памяти.

Элементарной ячейка хранения данных флэш-памяти представляет собой транзистор с плавающим затвором, особенность которого состоит в том, что он умеет удерживать электроны (заряд). Именно на различном методе соединения таких ячеек разработаны основные типы флэш-памяти — NOR и NAND.

Архитектура NOR получила название благодаря логической операции ИЛИ–НЕ (англ. NOT OR), а NAND И–НЕ (англ.NOT AND).  
 Если под операндами понимать значения ячеек памяти, то в архитектуре NOR единичное значение на битовой линии будет наблюдаться только в том случае, когда значение всех ячеек, подключенных к данной битовой линии, равно нулю т.е. все транзисторы закрыты (Рис.1.2.3.2). Тогда архитектура NAND подразумевает, что битовая линия имеет нулевое значение в случае, когда все подсоединенные к ней транзисторы открыты, и единичное значение – когда хотя бы один из транзисторов закрыт (Рис.1.2.3.1).

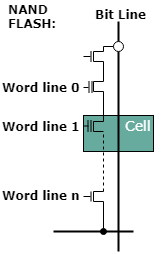
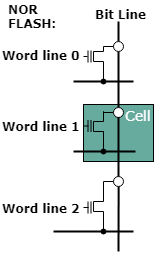
 

Рис. 1.2.3.1.NAND флэш-память Рис.1.2.3.2.NOR флэш-память

Требования хранения больших объемов и компактности однозначно определяют технологию используемой флэш-памяти. Кроме того, возможность выполнять в памяти записанный программный код — XIP Capability (XIP, eXecute In Place – мгновенная запись) является не менее важным критерием выбора.

Такая возможность существует у NOR–технологии и отсутствует у NAND. А плотность записи данных для технологии NAND порядком превосходят достигнутые в NOR.

Таким образом, основным назначением памяти, произведенной по технологии NAND, является хранение данных, а по технологии NOR — хранение исполнимого программного кода и, в меньшей степени, данных.

Более подробно о строении этих архитектур написано в источнике <https://ru.bmstu.wiki/%D0%A4%D0%BB%D0%B5%D1%88-%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C>

**Особенности флеш-памяти**

* Флэш-память может быть прочитана сколько угодно раз, но писать в такую память можно лишь ограниченное число раз (максимально - около миллиона циклов).
* Карты флэш-памяти не содержат в своем составе подвижных частей, что обеспечивает компактность и высокую сохранность данных при их использовании в мобильных устройствах. Кроме того, они обладают высокой плотностью упаковки (ячейки на 30 % меньше ячеек DRAM), низким потреблением, электрическим стиранием и записью.
* Флэш-память имеет особую технологию организации запоминающих ячеек. Каждая ячейка состоит всего из одного униполярного (полевого) транзистора. Разрядность данных внешнего интерфейса — 8 или 16 бит. Стертые (чистые) ячейки хранят единицу во всех битах. При программировании (записи) нужные биты обнуляются**. Стирание производится для всей матрицы ячеек (блоками или полностью всей микросхемы, стирание одиночной ячейки невозможно).** Это позволяет значительно увеличить производительность в режиме программирования (записи).
* Чтение флэш-памяти не отличается от чтения любой другой, а для программирования (записи) каждого байта (слова) приходится выполнять процедуру, состоящую из операций записи и считывания, направленных к микросхеме флэш-памяти. Так, получается что, **в устройстве с флэш-памятью легко предоставляется возможность перепрограммирования без извлечения микросхем из устройства**.
* При чтении время доступа у флэш-памяти составляет 35-200 нс. Стирание информации у микросхем середины 90-х годов занимало 1–2 секунды, программирование (запись) байта — порядка 10 мкс. Сейчас это время значительно сократилось. От ошибочного стирания (записи) предохраняют различные методы программной и аппаратной защиты. Под программной защитой представляется ключевая последовательность команд, в случае ее нарушения операции стирания и записи не начнутся. Аппаратная защита не допускает стирание и запись, если на определенные входы не поданы требуемые уровни напряжения.

**1.2.4. Оперативная память**

*Определение 1.2.8*

**Оперативная память** (оперативное запоминающее устройство – ОЗУ или *RAM, Random Access Memory*) — память с произвольным доступом, область временного хранения данных, обеспечивающая функционирование программного обеспечения, т.е. это совокупность ячеек для хранения данных – байтов, каждый из которых имеет свой адрес, позволяющий обратиться к любому байту памяти.

Назначение ОЗУ:

* Хранение, обработка данных и команд.
* Хранение результатов вычислений, произведенных процессором.
* Запись, считывание содержимого ячеек.

Особенности ОЗУ:

* Оперативная память является энергозависимой, иначе говоря при отключении питания все данные на ней удаляются.
* Оперативная память может напрямую прочитать/записать данные из любой ячейки ОЗУ, не затрагивая при этом остальные.

Память представляет собой совокупность ячеек для хранения данных – байтов (чаще всего 1 или 4 бит), каждый из которых имеет свой адрес, позволяющий обратиться к любому байту памяти.

**Типы оперативной памяти:**

Существует 2 типа памяти, возможных к применению в качестве ОЗУ в компьютере:

* *SRAM (Static random access memory* – память статического типа*)* — хранит записанные данные до тех пор, пока не появятся новые или не выключат питание;
* *DRAM (Dynamiс random access memory* – память динамического типа*)* — хранит данные лишь ограниченное время, после чего данные необходимо регенерировать, в противном случае, они будут потеряны. оперативная память это

**1.2.5. SRAM**

Главное преимущество статической памяти – высокая скорость чтения и записи. Это достигается благодаря минимальному времени доступа микросхемы SRAM (не больше 2 нс) и возможностью синхронно работать с процессорами. К достоинствам статической памяти также следует отнести низкое энергопотребление, простоту интерфейса и надежность в эксплуатации.  
Область применения — кэширование информации в процессоре и других устройствах (например, в регистрах процессора).

SRAM содержит совокупность триггеров ( – логические устройства, которые могут находиться в двух устойчивых состояниях: логический ноль и логическая единица), каждый из которых хранит 1 бит данных. Подобно ячейкам динамической памяти, триггеры объединяются в матрицу, которая состоит из строк (row), и столбцов (column), которые, в свою очередь, называются битами(bit).

Главный недостаток статической памяти – низкая плотность хранения данных (шесть–восемь элементов на бит, вместо двух у DRAM). Таким образом, микросхемы SRAM имеют бо́льшие габариты, из-за чего значительно возрастает стоимость килобайта памяти.

**Типы SRAM**

Существуют следующие типы статической памяти:

* Синхронная статическая память (одновременно производятся все операции только по тактовому сигналу, в связи с этим скорость обработки ограничивается тактовой частотой).

Именно на синхронной статической памяти реализуется кэш первого уровня современных процессоров.

* Асинхронная статическая память (чтение и запись выполняются по фронту сигнала данных, другими словами, контроль совершается поступающими данными).

Отличается быстрым доступом к информации.

**1.2.6. DRAM**

Динамическая память — оперативная память ПК, которая предоставляется программе в процессе ее работы за вычетом сегмента данных, стека и тела программы. Динамическое размещение данных определяет использование динамической памяти в процессе работы программы. При таком размещении заранее не известны ни тип, ни количество данных, также к ним невозможно обратиться по именам, как например, к статическим переменным.

DRAM обладает большой емкостью и стоимостью порядком ниже SRAM, но при этом она намного медленнее. Так как динамическая память изготавливается на основе конденсаторов небольшой емкости, быстро теряющих заряд, из-за чего во избежание потерь хранимых данных их нужно подзаряжать через определенные промежутки времени. Такой процесс называется регенерацией памяти и существенно тормозит работу системы поскольку в это время обмен данными с памятью невозможен. Регенерация осуществляется специальным контроллером, установленным либо на материнской плате, либо на кристалле центрального процессора.

**Виртуальная память компьютера.**

Метод управления памятью, отображающий используемые программами виртуальные адреса в физические адреса в памяти ПК.

Основная память является набором смежных сегментов.

Операционная система управляет виртуальными адресными пространствами и соотносит оперативную память с виртуальной. Программное обеспечение в операционной системе может расширить эти возможности, с целью обеспечения виртуального адресного пространства, которое может превысить объем оперативной памяти и ,следовательно, иметь больше памяти, чем есть в компьютере.

Виртуальная память позволяет изменять ресурсы памяти, делать объем оперативной памяти намного больше, для того чтобы пользователь, поместив туда как можно больше программ, сэкономил время и увеличил эффективность своего труда.

*Определение 2.1.15*

**Ассоциативная память** или **ассоциативное запоминающее устройство** — память, адресация в которой определяется содержанием объекта (автоматический поиск производится не по расположению данных), т.е. результатом поиска будут все адреса, где встречаются такие же данные.   
 Этот особый вид машинной памяти используется в приложениях очень быстрого поиска. специфически логическая память

*Определение 1.2.9*

Порт — это какое-либо средство вводы-вывода, отображенное в адресное пространство памяти.

По адресу порта вывода записываются данные, которые отправились через порт с определенного устройства вывода. По адресу порта ввода считываются данные, принятые с устройства ввода. Так осуществляется работа с некоторыми устройствами ввода-вывода(например, с клавиатурой, сетевой картой). Они вписаны в адресное пространство памяти, но при этом сами по себе не являются памятью, а только устройствами обмена информации.

Порты ввода-вывода соединяют два устройства. Однако часто целесообразно подключить к одному порту передачи данных несколько устройств, причем необязательно однотипных.

Системная шина (см. *Определение 1.1.5)* состоит из трех частей:

1. шина адреса, на которой устанавливается адрес требуемой ячейки памяти или устройства, с которым будет происходить обмен информацией;
2. шина данных, по которой, собственно, и будет передана необходимая информация;
3. шина управления, регулирующая этот процесс (например, один из сигналов на этой шине позволяет компьютеру различать между собой адреса памяти и устройств ввода/вывода).

Частный случай шины - шина с двумя сторонами, которая называется мост. Она удобна тем, что на ней нет адресации.

*Определение 1.2.10*

**Периферийные устройства** - аппаратура, которая позволяет вводить [информацию](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) в [компьютер](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) или выводить ее из него[.](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B8%D1%84%D0%B5%D1%80%D0%B8%D0%B9%D0%BD%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE#cite_note-1)

Взаимодействие периферийных устройств с адаптером происходит через порты ввода/вывода.

По способу передачи информации порты ввода-вывода делятся на:

Последовательные – информация передается по одному биту, последовательно бит за битом; для передачи информации используется один провод. Подключаются внешние устройства, находящиеся на расстоянии от компьютера.

Параллельные – несколько битов информации передается одновременно; для передачи информации используется несколько проводов. Подключаются устройства, находящиеся рядом с компьютером.

## 

## 1.2.7. Системы электропитания и энергосбережения

Наиболее важную часть средств вычислительной техники (СВТ) составляет источник питания, основное назначение которого — снабжать напряжением с заданными параметрами и качеством все узлы и составные блоки устройства вычислительной техники. Качество его функционирования в значительной степени определяет работу СВТ.

*Определение 1.2.12*

**Гибернация** — это технология, которая позволяет «выключить» компьютер при этом, не закрывая открытые приложения и не теряя не сохраненные результаты работы. После восстановления работы компьютера из гибернации, вы можете продолжить работу с того места, где вы закончили. При этом, находясь в режиме гибернации, компьютер не потребляет электроэнергию.

**Потребление энергии в разных режимах работы ЭВМ:**

* Обычный

Все компоненты компьютера потребляют столько энергии, сколько им нужно.

* Сон

Выключаются периферийные устройства, а на процессор, оперативную память, беспроводные модули и прочие компоненты в таком режиме подается минимальное напряжение. Возобновления работы с компьютером с прежнего места, происходит значительно быстрее, чем из режима гибернации.

* Гибернация

Компьютер обесточивается полностью (остается только питание на кварцевом генераторе для поддержания даты/времени) таким образом это самый энергосберегающий режим. При этом на жесткий диск записывается точная копия оперативной памяти компьютера в момент выключения и при включении компьютера этот образ восстанавливается с жесткого диска обратно в оперативную память. Иными словами, компьютер запоминает состояние системы перед уходом в режим гибернации, а затем восстанавливает все вкладки и настройки какие были.

**Охлаждение**

1. Система воздушного охлаждения
   1. Пассивная

*Составляющие системы:* радиатор.

*Принцип работы:* тепло от нагревающегося элемента передается на радиатор за счет теплопроводности материала или при помощи тепловых трубок, далее радиатор рассеивает тепло в окружающее пространство. Эффективность работы радиатора зависит от площади поверхности и материала изготовления.

*Достоинства:* экономность, надежная работа, безопасность, отсутствие шума.

*Недостатки:* низкая эффективность для современного оборудования.

* 1. Активная

*Составляющие системы:* вентилятор, очень часто для эффективности работы добавляют радиатор.

*Принцип работы:* горячий воздух отводится за приделы системного блока. Обычно устанавливают один или несколько вентиляторов, которые производят циркуляцию воздушного потока от передней стенки корпуса к задней.

*Достоинства:* низкая стоимость, простота в установке и обслуживании

*Недостатки:* основной источник шума в компьютере, низкие, по сравнению с другими системами, показатели эффективности.

2. Система жидкостного охлаждения

*Составляющие системы:* помпа, теплосъемник, радиатор, резервуар с рабочей жидкостью, тепловые трубки, датчик потока жидкости

*Принцип работы:* тепло передается от нагревающегося компонента к радиатору при помощи рабочей жидкости, которая циркулирует в системе. Обычно в качестве рабочей жидкости применяется дистиллированная вода с добавками, обладающими бактерицидный или антигальванический эффект.

*Достоинства:* почти бесшумная работа (слышен шум журчания воды), высокая эффективность охлаждения.

*Недостатки:* высокая стоимость, сложность установки, большой размер системы, высокая вероятность повреждения ряда ключевых компьютерных компонентов при выходе из строя.

1.3. Специализированные вычислители

*Определение 1.3.1*

**Специализированные вычислители —** компьютеры, предназначенные по архитектуре и набору команд под узкий класс задач или под конкретную задачу.

При проектировании, для соответствия устройства поставленным требованиям (техническим, экономическим – общая стоимость, складывающаяся из стоимости всех составляющих компьютера), приходится жертвовать характеристиками, такими как: габаритность компьютера, производительность, функциональность и надежность ради выгоды в других параметрах, таких как: надежность компьютера, стоимость устройства, энергопотребление и тепловыделение.

Пример: Графический процессор (англ. graphics processing unit, GPU) в видеокарте или специальные компьютеры повышенной надежности для вооруженных сил.

Одним из наиболее простых способов классифицировать различные типы вычислительных устройств является определение их способностей. Все вычислители могут, таким образом, быть отнесены к одному из трех типов:

1. специализированные устройства, умеющие выполнять только одну функцию;
2. устройства специального назначения, которые могут выполнять ограниченный диапазон функций;
3. устройства общего назначения, используемые сегодня. Название «компьютер» применяется, как правило, именно к машинам общего назначения.

*Определение 1.3.3*

**Специализированный процессор** — это процессор, у которого особенности архитектуры, набора структурных блоков, системы команд или конструктивно – технологического исполнения позволяют значительно повысить эффективность решения достаточно узкого круга специальных задач по сравнению с иными применениями.

Например, разработка специализированных процессоров для нейросетей. Также весьма сильно внедряется вид процессоров,называемый криптопроцессорами, которые автоматически защищают информацию, т.е ее запаковывают и распаковывают из защищенного вида.

Определение сопроцессора см. в главе 1.2.2.

Различают следующие виды сопроцессоров:

*математические сопроцессоры* общего назначения (для ускорения вычислений с плавающей запятой;

*сопроцессоры ввода-вывода*, разгружающие центральный процессор от контроля за операциями ввода-вывода или расширяющие стандартное адресное пространство процессора;

сопроцессоры для выполнения каких-либо узкоспециализированных вычислений.

Микропроцессор, оперативная память, флеш-память, порты ввода/вывода, таймеры, интерфейсы связи – все это заключено в микроконтроллере.

Наличие подобных устройств позволяет уменьшать нагрузку, которая приходится на центральный процессор. Благодаря этому значительно возрастает производительность системы. Механизм такой помощи заключается в том, что контроллеры освобождают компьютерные центральные процессоры от самых медленных операций ввода или вывода информации.

*Определение 2.1.18*

**Математический сопроцессор** (*англ. floating point unit, FPU*, сопроцессор с плавающей точкой) — это сопроцессор расширяющий командное множество центрального процессора, который обеспечивает его функциональностью модуля операций с плавающей точкой.

Принципиальные отличия математического сопроцессора от обычного процессора состоят в том, что он выполняет не математические операции, а операции для представления вещественных чисел с плавающей точкой и устроен через стек.

Модуль операций с плавающей точкой — это часть процессора для выполнения широкого спектра математических операций над вещественными числами.

Модуль операций с плавающей точкой поддерживает работу на уровне примитивов — загрузка, выгрузка вещественного числа или математические операции над ними, которые выполняются одной командой, за счет чего достигается значительное ускорение таких операций.

*Определение 1.3.6*

**Цифровой сигнальный процессор (ЦСП)** — это специализированный микропроцессор, архитектура которого оптимизирована для операций, необходимых для цифровой обработки сигналов в режиме реального времени. ЦСП измеряет, фильтрует и/или сжимает постоянный аналоговый сигнал.

Примеры использования ЦСП:

* Аудио-интерфейс компьютера (выдает в наушники звук, считывает звук с микрофона);
* Аналоговые устройства;
* В сотовом телефоне: из эфира ЦСП ловит радиоволну. ЦСП используется для того, чтобы из радиоволны выделить лишь ту часть, которая нужна для функционирования устройства. Как правило, их два: один занимается высокочастотной частью, второй занимается низкочастотной частью — приводит в порядок аудиосигнал.

Математически эти задачи сводятся к поэлементному перемножению элементов многомерных векторов действительных чисел, последующему суммированию произведений. Сигнальные процессоры оптимизированы по быстродействию для выполнения именно таких операций. ЦСП используют специальную архитектуру памяти, позволяющую получать данные и инструкции одновременно (гарвардская архитектура)*.*

*Определение 1.3.7*

**Потоковые процессоры** – это процессоры, в основе которых лежит принцип обработки многих данных с помощью одной команды. Эта технология позволяет выполнить одно и то же действие, например, вычисление и сложение, над несколькими наборами чисел одновременно.

Может быть отдельный потоковый процессор (Single-streaming processor - SSP) и многопотоковый процессор (Multi – Streaming Processor - MSP).

Ярким представителем потоковых процессоров является семейство процессоров intel, начиная с Pentium. Эта технология позволяет выполнить такие сложные и необходимые задачи как обработка речи, кодирование и декодирование видео- и аудиоданных, разработка трехмерной графики и обработка изображений.

*Определение 1.3.8*

**Графический процессор** (GPU – Graphics Processing Unit) — процессор, основной задачей которого является обработка графики и вычислений с плавающей точкой.

GPU облегчает работу главного процессора. Он создает графику, текстуры, цвета (т.е. создает изображение и выдает его пользователю на экран), а также занимается обработкой графики в формате 2D и 3D. Благодаря GPU компьютеру быстрее и легче удается выполнять важные задачи. Особенность графического процессора состоит в том, что он увеличивает скорость расчета графической информации на максимальном уровне. Его архитектура устроена так, что позволяет более эффективно обрабатывать визуальную информацию, чем центральный CPU компьютера.

*Определение 1.5.2*

**Аналого-цифровой преобразователь***(***АЦП**, *англ. Analog-to-digital converter, ADC*) — функция, преобразующая входной аналоговый сигнал, в дискретный код (цифровой номер).

*Определение 1.5.3*

**Эффективная разрядность** (*ENOB — Effective Number of Bits*) — параметр АЦП, характеризующий меру его динамического диапазона (точности). Однако существует несколько способов его расчета, которые приводят к разным результатам. Если неизвестно, по какой формуле она была вычислена, то этот параметр лучше не использовать, чтобы избежать ошибок.

**1.2.8.Графический процессор**

**Использование графических процессоров для вычислений** (взаимодействие графического и центрального процессоров).

У GPU нет средств прямого взаимодействия с устройствами ввода-вывода (помимо монитора), а также доступа к оперативной памяти компьютера. По этой причине управление графическим процессором осуществляется только посредством центрального процессора. Схема взаимодействия центрального и графического процессоров приведена на рис. 1.3.

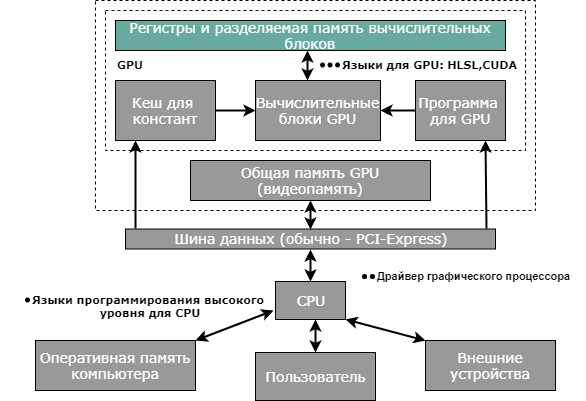


Рис. 1.2.8.1. Схема взаимодействия центрального и графического процессоров

Графические ускорители подключаются к системной плате пк посредством высокоскоростной шины данных (в настоящее время PCI-Express). С помощью этой шины CPU получает доступ к видеопамяти, а также к определенным разделам кэш-памяти, находящейся на самом графическом процессоре. Через эту же шину центральный процессор загружает в графический процессор программу и запускает ее.

Прежде, чем запустить программу, исполняемую на GPU, центральный процессор передает графическому данные двух типов:

* значения констант, используемых в программе;
* один либо несколько крупных массивов данных для потоковой обработки.

Т.к. к константам нужен постоянный и быстрый доступ, то они вносятся в кэш-память (или регистры), размещенную на кристалле GPU. Массивы данных зачастую бывают столь огромны, что полностью в кэш-память не помещаются. Но при простейшей потоковой обработке каждый из элементов массивов данных используется один единственный раз. Таким образом для хранения этих массивов предназначена видеопамять (общая память), которая представляет собой отдельные микросхемы на плате графического ускорителя. Она функционирует медленнее кэш-памяти и регистров, однако обладает большим объемом, до нескольких гигабайтов.

Результаты своей работы графический процессор может сразу записывать в раздел видеопамяти, который называется буфером кадра, откуда они передаются на монитор. Однако имеется также возможность вообще не показывать расчет на экране, а копировать результаты из видео-памяти в оперативную память компьютера, где они становятся доступными для последующей обработки центральным процессором. На этом и основано применение графических процессоров для вычислений общего назначения.

**Устройство GPU**

У графического процессора много ядер, функционирующих преимущественно на низких скоростях. Они занимаются вычислениями пикселей и вершин. Обработка последних в основном происходит в системе координат. GPU обрабатывает различные задачи, создавая на экране трехмерное пространство, то есть объекты в нем перемещаются. Каждый из типов графического процессора фильтрует треугольники, входящие в него. Он определяет, какие на виду, удаляет те, которые скрываются за другими объектами, прорисовывает источники света и определяет, каким образом эти источники влияют на цвет.

**Существуют два вида графических процессоров:**

1) Встроенный. Он есть практически во всех компьютерах. Его устанавливают в CPU, чтобы сделать потребление энергии в несколько раз ниже.

2) Дискретный. Его устанавливают на отдельные модули, так как отличается он своей мощью, но ему необходимо отличное охлаждение.

1.4 Вычислители с множественными потоками команд и одиночным потоком данных (MISD), истинно параллельные процессоры (MIMD), систолические вычислители

*Определение 1.4.1*

**MISD** (англ. Multiple Instruction stream, Single Data stream) — тип архитектуры параллельных вычислений, в котором несколько функциональных модулей (два или более) выполняют разные операции над одними данными.

*Определение 1.4.2*

**MIMD** (англ. Multiple Instruction stream, Multiple Data stream) — множественный поток команд и данных, который содержит некоторое число процессоров, которые асинхронно выполняют различные команды над различными данными. Т.е. несколько независимых процессоров работают как часть большой системы. Таких машин очень мало. В эту категорию попадают большинство параллельных процессоров. К MIMD-машинам относятся и мультипроцессоры (машины с общей памятью), и мультикомпьютеры (машины с обменом сообщениями).

*Определение 1.4.3*

**SIMD** (*Single Instruction Stream & Multiple Data Stream*) или ОКМД (Одиночный поток Команд и Множественный поток Данных) - архитектура, в которой есть возможность выполнять одну арифметическую операцию сразу над многими данными - элементами вектора.

*Определение 1.4.4*

**Систолические вычислительные системы**  — системы класса SIMD, основным принципом которых является то, что все данные регулярно и ритмически проходящие через массив, используются многократно. Это позволяет значительно повысить эффективность и достичь высокой вычислительной производительности за счет распараллеливания вычислений и сокращения обмена систолической системы с внешними устройствами.

В фон-неймановских машинах данные, считанные из памяти, однократно обрабатываются в процессорном элементе (ПЭ), после снова возвращаются в память. Идея систолической матрицы : организация вычисления производится таким образом, что данные на своем пути от считывания из памяти до возвращения обратно пропускались через как можно большее число ПЭ.

Проведём аналогию между структурами вычислительной системы и живого организма : памяти можно отвести роль сердца, множеству ПЭ - роль тканей, а поток данных рассматриваем как циркулирующую кровь. Отсюда и название *систолическая матрица* (систола - сокращение предсердий и желудочков сердца при котором кровь нагнетается в артерии). Систолические структуры эффективны при выполнении матричных вычислений, обработке сигналов, сортировке данных и т.д.

В качестве примера возьмем линейный массив для алгоритма матричного умножения. В основе схемы лежит прохождение двух потоков данных навстречу друг другу. Последовательные элементы каждого потока разделены одним тактовым периодом, для того чтобы каждый из них мог пересечься с любым элементом встречного потока. Вычисления выполняются параллельно в процессорных элементах, каждый из которых реализует один шаг в операции вычисления скалярного произведения (IPS, Inner Product Step) и носит название *IPS- элемента*.

Таким образом, *систолическая структура* - это однородная вычислительная среда из процессорных элементов, которая совмещает свойства конвейерной и матричной обработки и обладает свойствами:

* вычислительный процесс в систолических структурах есть непрерывная и регулярная передача данных от одного ПЭ к другому без запоминания промежуточных результатов вычисления;
* каждый элемент входных данных выбирается из памяти однократно и используется необходимое количество раз по алгоритму, ввод данных осуществляется в крайние ПЭ матрицы;
* образующие систолическую структуру ПЭ однотипны и каждый из них может быть менее универсальным, чем процессоры обычных многопроцессорных систем. Тип ПЭ выбирается в соответствии с назначением систолической матрицы и структурой пространственных связей (наиболее распространены процессорные элементы, ориентированные на умножение с накоплением);
* потоки данных и управляющих сигналов обладают регулярностью, что позволяет объединять ПЭ локальными связями минимальной длины;
* алгоритмы функционирования совмещают параллелизм с конвейерной обработкой данных;
* улучшение производительности матрицы осуществляется за счет добавления в нее определенного числа ПЭ, при этом коэффициент повышения производительности при этом линеен.

**Гибкие архитектуры и дилемма «отказоустойчивость-эффективность»**

Рассмотрим несколько примеров гибких архитектур.

Машины сгруппированы в самостоятельный блок, на котором стоит процессор, память и все, что нужно для общения с шиной. Из внешних контроллеров есть только контроллер шины. Существует большая вероятность того, что за час что-то выйдет из строя. Но пользователь, по возможности, этого заметить не должен. Есть встроенные аппаратные средства диагностики, которые позволяют выполнять сканирование системы, исследуя аппаратные компоненты и программные конфигурации на наличие и устранение проблем. В это же время от аппаратных средств в ОС идет сигнал о выходе из строя одного из вычислителей. ОС анализирует какие задания этот вычислитель не успел посчитать (хотя получил). Эти задания до их исполнения хранятся в этой иерархии серверов, распределяющих задачи. Затем, чтобы примерно выровнять нагрузку на все компьютеры, планировщик распределяет эти задачи. Таким образом пользователь особо ничего не заподозрит, несмотря на то, что задержка будет более чем двойная.

Другой пример, облачные системы – системы распределенного способа обработки данных. Google позволяет через интернет обучать нейросети, используя их удаленные процессоры. Это система, которая распределенно хранит информацию в облаке и, кроме того, ее обрабатывает. Она автоматически решает вопрос, какие данные использовать. В том числе гибкость достигается и этим.

Также, архитектура систем сотовых связей тоже есть гибкая архитектура. Система сотовой связи допускает большое разнообразие по вариантам конфигураций и по набору выполняемых функций. Сложность и гибкость системы можно понять из того факта, что она может обеспечивать передачу не только речи, но и других видов информации, например, текстовых сообщений и компьютерных данных. В части передачи речи может быть реализована обычная двусторонняя телефонная связь, многосторонняя телефонная связь, голосовая почта. При организации обычного двустороннего телефонного разговора, который начинается с вызова, возможны режимы автодозвона, ожидания вызова и переадресации вызова.

Также рассмотрим **дилемму «отказоустойчивость-эффективность»**.

Где ответственность за ошибку ниже, используется динамическое балансирование: когда имеется возможность при выходе какой-либо взаимозаменяемой части ресурса, заменить другими с общим уменьшением производительности. Т.е. есть статическое в этом смысле распараллеливание, есть динамическая балансировка нагрузки.

Где система жесткая, т.е. вероятность ошибки должна стремиться к нулю, используется статическое распараллеливание. Кроме того, при таком решении задачи центральный компьютер может считать вероятность того, что найденное решение правильное.

1.5. Разрядность элементов вычислителя

*Определение 1.5.1*

**Разрядность процессора** — это число битов, обрабатываемых процессором одновременно, т.е. скорость. Процессор может быть разных разрядов: 8-, 16-, 32- и 64-разрядным и т.д. (разрядность с 128 по 512 — это возможные разрядности шины памяти видеокарт). Вместе с быстродействием, разрядность характеризует объем информации, перерабатываемый процессором за определенный промежуток времени. Разрядность вычислителя зависит от разрядности шин. Современные технологии пытаются прийти к концепции: шина быстрее, чем процессор.

Чем больше разрядность, тем медленнее, но точнее процессор будет считать. Эффективность зависит от поставленной задачи.

*Определение 1.5.6*

**AVX** (*Advanced Vector Extensions*) *—* новый набор SIMD инструкций, функций и схем кодирования для x86-процессоров Intel, который повышает производительность рабочих нагрузок.

Улучшения:

* Ширина векторных регистров SIMD (обеспечивает параллелизм на уровне данных) увеличивается с 128 до 256 бит. Существующие 128-битные SSE-инструкции будут использовать младшую половину новых YMM регистров, не изменяя старшую часть. Для работы с YMM-регистрами добавлены новые 256-битные AVX-инструкции. Возможно расширение векторных регистров SIMD до 1024 бит, на данный момент реализовано 512-битные версии.
* Неразрушающие операции. Набор AVX-инструкций использует трехоперандный синтаксис. К примеру, вместо m=n+m можно использовать s=n+m, при этом регистр m – неизмененный. Когда значение m используется дальше в вычислениях, это избавляет от необходимости сохранять перед вычислением и восстанавливать после вычисления регистр, содержавший m, из другого регистра или памяти.

AVX – очень длинный регистр, в который можно положить одно 512-разрядное число. Также, можно положить и два числа независимо работающих, каждое из которых 256-разрядное. Помножив два таких регистра, получится заготовка для скалярного произведения. Далее можно разделить еще пополам. Как итог, четыре 128-разрядных числа. Делить можно вплоть до char (8 бит), тогда получится 64 числа — столько элементов размером 1 char можно хранить в одном регистре, причем регистр будет работать как вектор. К тому же, операция, которая выполняется над этим регистром, выполняется очень быстро (в один машинный такт) независимо над каждым кусочком. Это значительно повышает производительность для интенсивных вычислений с плавающей точкой.  
AVX также специализируется на работе с числами с плавающей точкой типа float и double.

[**Понятие о масштабируемой разрядности**](#2dlolyb)**.**

Есть регистры α, β, γ, δ каждый — 32 бита, если обычные типы long и unsigned, то мы этими регистрами оперируем независимо, но оказывается возможным оперировать парами регистров, когда они выстроены друг за другом. Регистры полноценно работают с 64 разрядом, а есть такие, которые работают по четверкам — α, β, γ, δ (соотв. разрядность 128, Рис.1.3.3). Как итог: масштабируемая разрядность это — использование «раздельных» регистров вместе — над парами или четверками осуществляется одна и та же операция, понимая, что справа — младшие разряды, а слева — старшие.

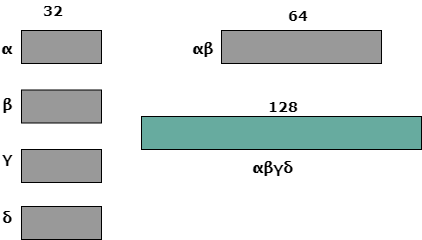


Рис. 1.5.1. Масштабируемая разрядность

1.6. Программная среда специализированного вычислителя

*Определение 1.6.1*

**BIOS (частный случай)** (*Basic Input Output System*) *—* базовая система ввода-вывода; набор микропрограмм, реализующих [API](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) для работы с аппаратурой компьютера и подключенными к нему устройствами. В эту систему входят разные программы ввода-вывода, обеспечивающие связь между ОС, прикладными программами с одной стороны и устройствами, входящими в состав компьютера (внутренними и внешними) с другой.

СуществуетBIOS для материнских плат и для периферийных устройств.

На сегодняшний день BIOS это сложная система, которая состоит из значительного числа утилит, предназначенных для автоматического распознавания, установленного на компьютер оборудования, его настройки и контроля функционирования. Вызов программ BIOS обычно осуществляется через программные или аппаратные прерывания. При каждом включении питания компьютера типа IBM PC (или совместимого с ним) и до начала загрузки ОС процессор компьютера осуществляет процедуру BIOS под названием "Самотест по включению питания" – POST (Power On Self Test). Главной целью процедуры POST является проверка базовых функций и подсистем компьютера (таких как память, процессор, материнская плата, видеоконтроллер, клавиатура, гибкий и жесткий диски и т. д.) перед загрузкой ОС. Это в некоторой степени застраховывает пользователя от попытки работать на поврежденной системе, что могло бы послужить причиной, например, разрушения пользовательских данных на HDD.

BIOS представляется как микросхема, которая установлена на материнской плате компьютера. Обратим внимание, что название ROM BIOS сейчас не совсем справедливо, так как "ROM" подразумевает применение постоянных запоминающих устройств (Read Only Memory), а для хранения кодов BIOS в настоящее время используют в основном перепрограммируемые запоминающие устройства. Более перспективной для хранения системы BIOS считается флэш-память. Она дает возможность изменять функции для поддержки новых устройств, подключаемых к компьютеру.

*Определение 1.6.2*

**Firmware** (рус. прошивка) *—* специализированное программное обеспечение внутри техники с микроконтроллером. Практически все электронные устройства помимо простейших включают в себя Firmware: компьютерные мыши, клавиатуры, жесткие диски, маршрутизаторы, современные модели автомобилей, телевизор, стиральная машина и т.д.

*Определение 1.6.3*

**Операционные системы реального времени (**[**ОСРВ**](http://embedded.prosoft.ru/tags/osrv/) **(real-time operating system - RTOS))** — операционные системы, которые способны предоставить предсказуемое время обработки неожиданно появляющихся внешних событий. Разделяют ОС «жесткого» и «мягкого» реального времени: для первых временные характеристики гарантированы, и выход за их границы расценивается как отказ, для вторых временные ограничения обычно соблюдаются, и выход за их пределы является снижением производительности. Основная масса современных ОСРВ представляют собой встраиваемые ОС.

Окружение времени выполнения (run-time environment) программных модулей - RTE. Существуют обычные ОС, если их упрощать и убирать многие функции, то получается ОСРВ. При убирании почти всех функций получается RTE. RTE - небольшое количество функций, которые уже не являются операционной системой и часто стоят на различных специфичных ЭВС. Могут включать в себя до пары десятков функций. ОСРВ, в отличие от RTE, может иметь какую-то вариативность в действиях, другими словами, универсальность.

*Определение 1.6.4*

**Встраиваемые операционные системы** — ОС, которые предназначены для управления специализированными устройствами и, как следствие, умеющие работать в условиях ограниченных ресурсов (небольшие объемы памяти, нехватка вычислительных мощностей и т.п.) и в необслуживаемом режиме. Отличительными свойствами встраиваемых ОС считаются модульная структура, компактность, производительность, масштабируемость и повышенная отказоустойчивость.

ОСРВ и RTE отличаются от обычных ОС областью применения, функциональностью, скоростью работы и отказоустойчивостью. ОСРВ, к примеру, могут не обладать возможностью воспроизводить звук (и кучи других функций), однако могут работать быстрее и надежней при обстоятельствах, в которых ОС общего назначения затормозит. Подобный переход от ОСРВ к RTE.

Таблица 1.6.1

|  |  |
| --- | --- |
| ОС (OS *—* operating system) | Обычная система.  Может использоваться на компьютерах, ноутбуках, некоторых планшетах (Windows, Ubuntu, Android). |
| ОСРВ (RTOS *—* real time OS) | «Обкусанная» ОС, из которой выброшены все ненужные функции, но при этом сохраняется определенная универсальность (свобода действий, возможность решения широкого круга задач)  Пример использования: мосты, шлюзы, маршрутизаторы (RTOS Linux). |
| ОВВ (RTE *—* run-time environment) | Это уже нельзя назвать операционной системой, так как она «обкусана» до такой степени, что может решать только строго определенный узкий круг задач. Нет динамического выделения памяти, есть только стек, нет кэша. Нет периферических устройств, кроме портов: с одного мы всегда берем, на другой отдаем. Там нет никаких функций отладки.  Используется, например, в GPS-навигации, сотовых станциях. |

**Глава 2. Процессор (компьютер)**

**2.1. Узлы процессора**

*Определение 2.1.*

**Процессор общего назначения** — такие процессоры могут достаточно эффективно решать широкий класс задач управления, вычислительных и прочих. Именно процессоры этого класса используются в качестве центрального процессора в настольных рабочих станциях.

Задача **центрального процессора** *—* это выполнение программ, которые находятся в основной памяти.

*Определение 2.1.1*

**Регистры** *—* это специальные ячейки памяти, расположенные непосредственно в процессоре. Работа с регистрами выполняется намного быстрее, чем с ячейками оперативной памяти, поэтому регистры активно используются как в программах на языке ассемблера, так и компиляторами языков высокого уровня. Они позволяют управлять выполнением программы, хранить временные результаты и т.д.

Доступ к значениям, хранящимся в регистрах, как правило, в несколько раз быстрее, чем доступ к ячейкам оперативной памяти, но объем оперативной памяти намного превосходит суммарный объем регистров

**Свойства регистров:**

* Регистры нужны для вычислений и связи ЦП с внешним миром.
* Большая часть кода программ состоит из команд копирования значений из оперативной памяти в регистры и обратно.
* Во всех операциях программы без регистров не обойтись.
* И если даже данные не хранятся в регистрах, то в них обязательно будут указатели на эти данные (адреса данных в памяти), других способов работы с данными у процессора нет, только через собственные регистры. А если речь идет исключительно о сторонних устройствах, то мы тоже используем регистры, только это уже не регистры процессора, а, например, регистры , SATA и так далее.
* К доступным регистрам пользователь может обращаться с помощью команд машинного языка. К этим регистрам, как правило, имеют доступ все программы — как приложения, так и системные. Обычно среди доступных регистров есть регистры данных, адресные регистры и регистры кода условия.

Регистры классифицируются на аккумуляторы, регистры данных, адресные регистры, индексные, регистры общего назначения, регистры команд и флаговые регистры.

*Определение 2.1.2*

**Аккумулятор** *—* главный регистр процессора над которым осуществляется подавляющее большинство арифметических и логических побитовых операций.

*Определение 2.1.3*

**Регистры данных** *—*регистры, которые используются для записи и хранения промежуточных вычислений.

*Определение 2.1.4*

**Адресные регистры** нужны для хранения адреса (или его части) операнда выполняемой программы, они формируют адреса данных и команд.Регистр адреса памяти содержит двоичное число – адрес области памяти. Выход этого регистра называется адресной шиной.

*Определение 2.1.5*

**Индексный регистр** используется в обычном режиме адресации, когда адресом является результат сложения содержимого индексного и базового регистра.

При сегментной адресации память делится на сегменты, состоящие из разного количества машинных слов. Адрес ячейки памяти складывается из адреса сегмента и смещения относительно начала сегмента. При таком режиме адресации базовый адрес сегмента (его начало) хранится в одном из регистров. Таких регистров может быть несколько; например, один — для операционной системы (т.е. использующийся при выполнении процессором кода операционной системы), другие — для приложений, выполняющихся в данный момент.

**Регистры общего назначения** используются для запоминания данных и/или операндов при исполнении команд.

Для примера рассмотрим регистры для процессоров Intel.

Регистры EAX, EBX, ECX, EDX – регистры общего назначения. Они имеют определенное назначение, но в них можно хранить какую угодно информацию.

Регистры EBP, ESP, ESI, EDI – это тоже регистры общего назначения. Они имеют уже более конкретное назначение. В них также можно хранить пользовательские данные, но делать это нужно уже более осторожно, чтобы не получить «неожиданный» результат.

*Определение 2.1.6*

**Счетчик команд** *—* специальный регистр, в котором хранится адрес команды, которая должна быть выполнена после того, как выполнится текущая команда.

*Определение 2.1.7(в главу прерывания)*

**Флаг** *—* это один или несколько битов памяти, принимающие двоичные значения (или комбинации значений) и характеризующие состояние какого-нибудь объекта. Флаг может принимать 0 (сброшен) или 1(установлен).

**Регистр флагов** носит название **FLAGS**. Это 32-разрядный регистр. Но при работе в защищенном режиме используются только старшие 16 разрядов. К младшим 16 разрядам этого регистра можно обращаться как к отдельному регистру с именем FLAGS.

Регистр используется побитно, т.е. каждый бит имеет определенное функциональное назначение, называющееся флагом. Они управляются различными командами для индикации состояния операции. Во всех случаях флаги сохраняют свое значение пока другая команда не изменит его.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| CF | 1 | PF | 0 | AF | 0 | ZF | SF | TF | IF | DF | OF | IOPL | | NT | 0 |

Таблица 2. 1. Флаговый регистр flags.

***CF (Carry Flag)*** – флаг переноса. Содержит значение "переносов" (0 или 1). Если результат предыдущей операции не уместился в приемнике и произошел перенос из старшего бита или если требуется заём (при вычитании), устанавливается в 1, иначе - в 0.

***PF (Parity Flag)*** – флаг четности. Проверяет младший байт результатов предыдущей команды. Нечетное число бит приводит к установке этого флага в 0, а четное - в 1.

***AF (Auxiliary Carry Flag)*** – дополнительный флаг переноса. Устанавливается в 1, если арифметическая операция приводит к переносу четвертого справа бита (бит номер 3) в регистровой однобайтовой команде. Данный флаг имеет отношение к арифметическим операциям над символами кода ASCII и к десятичным упакованным полям.

***ZF (Zero Flag)*** – флаг нуля. Устанавливается в качестве результата арифметических команд и команд сравнения. Если результат предыдущей команды равен 0, устанавливается 1.

***SF (Sign Flag)*** – знаковый флаг. Устанавливается в соответствии со знаком результата (старшего бита) после арифметических операций: положительный результат устанавливает 0, а отрицательный - 1.

***TF (Trap Flag)*** – флаг трассировки (пошагового выполнения). Если этот флаг установлен в 1, то процессор переходит в режим пошагового выполнения команд или управление временно передается отладчику.

***IF (Interrupt Flag)*** – флаг прерывания. При установлении состояния 0 прерывания запрещены, при 1 – разрешены.

***DF (Direction Flag)*** – флаг направления. Используется в строковых операциях для определения направления передачи данных. При нулевом состоянии строки будут обрабатываться в сторону уменьшения адресов, при нулевом – в сторону уменьшения адресов.

***OF (Overflow Flag)*** – флаг переполнения. Фиксирует арифметическое переполнение, т.е. перенос вниз старшего (знакового) бита при знаковых арифметических операциях.  
Например, если при сложении двух положительных чисел получается число со старшим битом, равным единице, то есть отрицательное. И наоборот.

***IOPL (Input/Output Privilege Level)*** – флаг уровня приоритета ввода-вывода. Используется в защищенном режиме работы процессора для контроля доступа к командам ввода-вывода в зависимости от приоритетной задачи.

***NT (Nested Task)*** – флаг вложенности задач. Используется в защищенном режиме работы процессор, фиксируя, что одна задача вложена в другую.

*Определение 2.1.8*

**Регистр указателя стека** *—* **это** *16-разрядный* регистр, который указывает на положения данных в стеке.

При стековой адресации выделяется специальный регистр, размещающий указатель на вершину стека. Такой режим адресации позволяет использовать некоторые команды, в которых отсутствует поле адреса.

В состав всех процессоров входит регистр или набор регистров, известный под названием **регистра слова состояния программы** (program status word — PSW) содержащий коды условий и другую информация о состоянии:

|  |  |  |
| --- | --- | --- |
| Символ | Позиция | Имя и назначение |
| P | PSW.0 | Флаг паритета. Устанавливается и сбрасывается аппаратуры в каждом цикле команды и фиксирует нечетное/четное число единичных бит в аккумуляторе, т.е. выполняет контроль по четности |
| - | PSW.1 | Не используется |
| OV | PSW.2 | Флаг переполнения. Устанавливается и сбрасывается аппаратно при выполнении арифметических операций |
| RS0 | PSW.3 | программой для выбора рабочего банка регистров |
| RS1 | PSW.4 | Выбор банка регистров. Устанавливается и сбрасывается |
| F0 | PSW.5 | Флаг 0. Может быть установлен, сброшен или проверен программой как флаг, специфицируемый пользователем |
| АС | PSW.6 | Флаг вспомогательного переноса. Устанавливается и сбрасывается только аппаратурными средствами при выполнении команд сложения и вычитания и сигнализирует о переносе или заеме в бите 3 |
| С | PSW.7 | Флаг переноса. Устанавливается и сбрасывается аппаратурными средствами или программой при выполнении арифметических и логических операций |

Таблица 2.2 Формат слова состояния программы (PSW)

Флаг переноса наиболее "активный" флаг PSW, который принимает участие и модифицируется в процессе выполнения множества операций, включая сложение, вычитание и сдвиги. Помимо этого, флаг переноса (С) выполняет функции "булевого аккумулятора" в командах, манипулирующих с битами. Флаг переполнения (OV) фиксирует арифметическое переполнение при операциях над целыми числами со знаком и делает возможным использование арифметики в дополнительных кодах. АЛУ не управляет флагами селекции банка регистров (RS0, RS1), и их значение полностью определяется прикладной программой и используется для выбора одного из четырех регистровых банков.

*Определение 2.1.9*

**Коды условий** *—* это последовательность битов, которые устанавливаются или сбрасываются процессором зависящие от результата выполненных операций.

*Определение 2.1.11*

**SSE** (*Streaming SIMD Extensions*)*—*ранняя версия AVX, который полностью ее заменяет, поэтому в новых разработках SSE не используется. MMX *(Multimedia Extensions)* это ранняя (20-тилетней давности) попытка сделать расширенный набор команд, в том числе и для векторов. **SSE – ее развитие, а** AVX **это современная сегодняшняя версия (**AVX2 или AVX512**)»**

**AVX** расширяет 128-битные регистры **SSE** до 256 бит(ymm0 - ymm15), при этом 128-битные SSE регистры ссылаются на младшие 128 бит соответствующего **AVX** регистра.

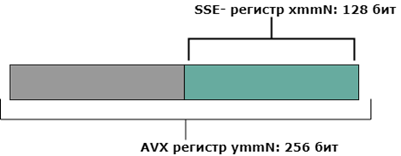


Рис. 2.1.1. AVX и SSE регистры

Используется для интенсивных вычислений, увеличивает производительность с вещественными числами и там, где высокая степень параллелизма.

Большинство компьютеров имеет как **внутренние**, так и **внешние** шины.

*Определение 2.1.12*

**Внутренняя шина** **– это** шина, подключающая все внутренние компоненты компьютера к материнской плате. Данный тип шин имеет название локальной шиной, служащих для подключения локальных устройств.

*Определение 2.1.13*

**Внешняя шина** **– это** шина, подключающая внешнюю периферию к материнской плате.

Однако, не всегда можно определить тип шины: например, шина SATA **–** внешняя или внутренняя? Диск можно подключить как снаружи, так и внутри. Аналогично, шина USB. Также стоит отметить, на материнских платах довольно-таки много внутренних usb-разъемов, к которым подключаются внутренние устройства, например, картридеры. Устройство внутри, но шина внешняя. Таким образом, шины SATA и USB нельзя назвать определенно внутренними или определенно внешними.

**Как работает шина:**

Некоторые устройства, которые соединены с шиной, являются активными и способны инициировать передачу информации по шине, другие - пассивные и ожидают запросов. Активное устройство – это задающие, пассивное – это подчиненное.

**Внешние формирователи**: Большая часть задающих устройств связаны с шиной через микросхему, называемую **драйвером шины** и являющуюся цифровым усилителем. Большая часть подчиненных устройств связаны с шиной **приемником шины**.

Для устройств, которые и задающие, и подчиненные устройством применяется **приемопередатчик шины** (трансивер).

Регистры вместе с АЛУ формируют тракт данных, с поступающими данными. Базовая операция тракта данных выполняется так: выбирается один или два регистра, АЛУ производит над ними операцию, после результат вновь помещается в какой-либо регистр.

АЛУ может выполнять следующие операции:

* логические операции.
* арифметические операции.
* операции циклического сдвига, инвертирования, сброса

Арифметико-логическое устройство функционально можно разделить на две части (Рис 2.1.1):

а) микропрограммное устройство, которое задает последовательность микрокоманд;

б) операционное устройство, реализующие заданную последовательность микрокоманд.

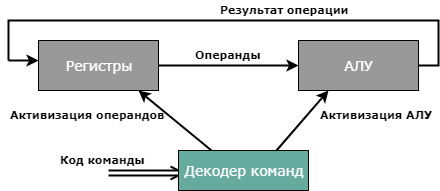


Рис 2.1.2 Структура АЛУ

Особенность АЛУ это его способность оперировать не только байтами, но и битами. Программно-доступные биты могут быть установлены, инвертированы, переданы, сброшены, проверены и использованы в логических операциях. Данная способность важна, ведь при управлении объектами часто применяются алгоритмы, которые содержат операции над выходными булевыми переменными, реализация которых средствами обычных микропроцессоров сопряжена с определенными трудностями.

**Процедура выполнения команд.**

Программа состоит из машинных команд, которая загружается в оперативную память компьютера. Для реализации программы процессор выполняет машинные команды в той последовательности, в какой они записаны в программе.

Для того чтобы процессор знал, какую команду нужно выполнять в определенный момент, существует *счетчик команд* – специальный регистр, хранящий адрес команды, которая должна быть выполнена после выполнения текущей команды. То есть, при запуске программы в этом регистре хранится адрес первой команды.

Счетчик команд работает с памятью, которая находится внутри процессора. Эта память носит название *очередь команд*, куда помещается одна или несколько команд непосредственно перед их выполнением. Таким образом, в счетчике команд хранится адрес команды в очереди команд, а не адрес оперативной памяти.

*Цикл выполнения команды* – это последовательность действий, которая совершается процессором при выполнении одной машинной команды. При выполнении каждой машинной команды процессор должен выполнить: *выборку, декодирование и выполнение.* Если в команде используется операнд, находящийся в оперативной памяти, то процессору придется выполнить еще две операции: *выборку операнда* из памяти и *запись результата* в память. Ниже описаны эти пять операций.

* *Выборка команды*. Блок управления извлекает команду из памяти (из очереди команд), копирует ее во внутреннюю память процессора и увеличивает значение счетчика команд на длину этой команды (разные команды могут иметь разный размер).
* *Декодирование команды.* Блок управления определяет тип выполняемой команды, пересылает указанные в ней операнды в АЛУ и генерирует электрические сигналы управления АЛУ, которые соответствуют типу выполняемой операции.
* *Выборка операндов.* Если в команде используется операнд, расположенный в оперативной памяти, то блок управления начинает операцию по его выборке из памяти.
* *Выполнение команды.* АЛУ выполняет указанную в команде операцию, сохраняет полученный результат в заданном месте и обновляет состояние флагов, по значению которых программа может судить о результате выполнения команды.
* *Запись результата в память.* Если результат выполнения команды должен быть сохранен в памяти, блок управления начинает операцию сохранения данных в памяти.

**Обобщая сказанное, составим цикл выполнения команды:**

1. Выбрать из очереди команд команду, на которую указывает счетчик команд.
2. Определить адрес следующей команды в очереди команд и записать адрес следующей команды в счетчик команд.
3. Декодировать команду.
4. Если в команде есть операнды, находящиеся в памяти, то выбрать операнды.
5. Выполнить команду и установить флаги.
6. Записать результат в память (по необходимости).
7. Начать выполнение следующей команды с п.1.

Программа, которая имитирует работу центрального процессора, называется **интерпретатором**.

**Устройство управления** (*см. определение 1.1.6)*

Для обеспечения автоматического вычисления по программе, процессор должен выполнять ряд дополнительных действий:

* извлечение из памяти очередную команду;
* расшифровывание ее и преобразование в последовательность стандартных действий;
* занесение в АЛУ исходных данных;
* сохранение полученных в АЛУ результатов;
* обеспечение синхронности работы всех узлов машины.

Для выполнения этих функций служит **устройство управления (УУ)**

УУ содержит важные регистры для хранения информации, необходимой в ходе выполнения текущей команды.

Регистр команды – служит размещением текущей команды, находящийся в нем в течение текущего цикла процессора.

Имеются регистры, которые содержат адрес команды, адреса операндов, счетчик адреса команды, операнды и результаты выполнения команды.

*Определение 2.1.16*

**Очередь** — это структура, которая разделяет поток данных где чтение слов происходит в порядке их записи.

*Определение 2.1.17*

**FIFO** (*first-in, first-out:* первым пришел, первым вышел) — отличается от очереди тем, что не может иметь приоритетов.

**Блок предсказания переходов**

Эффективность алгоритмов предсказания переходов критична для архитектур, в которых используется высокий уровень параллелизма инструкций. Такие переходы разрывают параллелизм, ведь необходимо ждать результат предыдущей инструкции, перед продолжением выполнений потока инструкций. Предсказание переходов определяет возможность того, какая будет взята инструкция, и если предсказание верно, то вычисляется дальнейший адрес выполнения.

Требуется Branch Target Buffer, сохраняющий результат ветвлений по мере продолжения выполнения кода, так к массиву прилагается алгоритм определений результатов следующего ветвления.

Декодеры преобразуют команды в макрокоманды, являющимися совокупностью команд, встречающихся в программах, под управлением которых в процессоре выполняются элементарные операции.

Простая инструкция преобразуется в 1–2 микрокоманды, являющимися совокупностью кодов микроопераций, которые управляются схемами в течение одного машинного такта.

Сложная инструкция из памяти микрокода выбирает последовательность микрокоманд, содержащая более двух микрокоманд. Используя технологию macrofusion, четыре декодера способны обработать одновременно пять команд, преобразуя их в четыре микрокоманды. Данная технология заключается в том, что ряд пар связанных между собой последовательных инструкций, представляются декодером одной микрокомандой. Таким образом, технология macrofusion позволяет закодировать одной микрооперацией две команды. В то время, как технология microfusion позволяет закодировать одной микрооперацией две операции, чтобы снизить нагрузку на конвейер для некоторых сложных команд.

Помимо основных узлов процессора необходимы и другие вспомогательные узлы, которые нужны для обработки данных. рассмотрим некоторые дополнительные возможности процессора:

*Определение 1.5.8*

**Упреждающая выборка команд** (*англ. instruction prefetch*)  *—* это метод, используемый компьютерными процессорами для повышения производительности путем извлечения инструкций или данных из их исходного хранилища в более медленной памяти в более быструю локальную память.

*Определение 1.5.9.*

**Контроллер прямого доступа к памяти** *—* (*англ. Direct Memory Access - DMA*) *—* блок, это которая позволяет некоторым аппаратным подсистемам получать доступ к основной системной памяти ( оперативная память ) независимо от центрального процессора (ЦП).

2.2. Ядро процессора

*Определение 2.2.1*

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

Ядро процессора состоит из нескольких функциональных блоков:

* блока выборки инструкций;
* блоков декодирования инструкций;
* блоков выборки данных;
* управляющего блока;
* блоков выполнения инструкций;
* блоков сохранения результатов;
* блока работы с прерываниями;
* ПЗУ, содержащего микрокод;
* набора регистров;
* счетчика команд.

**Блок выборки инструкций** осуществляет считывание инструкций по адресу, указанному в счетчике команд. Обычно, за такт он считывает несколько инструкций. Количество считываемых инструкций обусловлено количеством блоков декодирования, так как необходимо на каждом такте работы максимально загрузить блоки декодирования. Для того, чтобы блок выборки инструкций работал оптимально, в ядре процессора имеется предсказатель переходов.

**Предсказатель переходов** пытается определить, какая последовательность команд будет выполняться после совершения перехода. Это необходимо, чтобы после условного перехода максимально нагрузить конвейер ядра процессора.

**Блоки декодирования**, как понятно из названия, – это блоки, которые занимаются декодированием инструкций, т.е. определяют, что надо сделать процессору, и какие дополнительные данные нужны для выполнения инструкции. Задача эта для большинства современных коммерческих процессоров, построенных на базе концепции CISC, – очень сложная. Дело в том, что длина инструкций и количество операндов – нефиксированные, и это сильно усложняет жизнь разработчикам процессоров и делает процесс декодирования нетривиальной задачей.

**Блоки выборки данных** осуществляют выборку данных из кэш-памяти или ОЗУ, необходимых для выполнения текущих инструкций. Обычно, каждое процессорное ядро содержит несколько блоков выборки данных. Например, в процессорах Intel Core используется по два блока выборки данных для каждого ядра.

**Управляющий блок** на основании декодированных инструкций управляет работой блоков выполнения инструкций, распределяет нагрузку между ними, обеспечивает своевременное и верное выполнение инструкций. Это один из наиболее важных блоков ядра процессора.

**Блоки выполнения инструкций** включают в себя несколько разнотипных блоков:

**АЛУ** – арифметико-логическое устройство *(см. определение 1.1.3)*.

**FPU** (см.*определение 2.1.18)*

**Блок сохранения результатов** обеспечивает запись результата выполнения инструкции в ОЗУ по адресу, указанному в обрабатываемой инструкции.

**Блок работы с прерываниями.**

Работа с прерываниями – одна из важнейших задач процессора, позволяющая ему своевременно реагировать на события, прерывать ход работы программы и выполнять требуемые от него действия. Благодаря наличию прерываний, процессор способен к псевдопараллельной работе, т.е. к, так называемой, многозадачности.прерывание

Обработка прерываний происходит следующим образом. Процессор перед началом каждого цикла работы проверяет наличие запроса на прерывание. Если есть прерывание для обработки, процессор сохраняет в стек адрес инструкции, которую он должен был выполнить, и данные, полученные после выполнения последней инструкции, и переходит к выполнению функции обработки прерывания.

После окончания выполнения функции обработки прерывания, из стека считываются сохраненные в него данные, и процессор возобновляет выполнение остановленной задачи.

*Определение 2.2.4*

**Счетчик команд** – регистр, содержащий адрес команды, которую процессор начинает выполнять на следующем такте работы.

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, предназначенной 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.

*Определение 2.2.5*

**Многоядерный процессор** — центральный процессор, содержащий два и более вычислительных ядра на одном процессорном кристалле или в одном корпусе.

В многоядерных процессорах тактовая частота, как правило, намеренно снижена. Это позволяет уменьшить энергопотребление процессора без потери производительности. В некоторых процессорах тактовая частота каждого ядра может меняться в зависимости от его индивидуальной нагрузки. Ядро является полноценным микропроцессором, использующим все достижения микропроцессорной техники: конвейеры, внеочередное исполнение кода, многоуровневый кэш, поддержка векторных команд.

Каждое ядро также может использовать технологию SMT для поочередного исполнения нескольких потоков, создавая иллюзию нескольких «логических процессоров» на основе каждого ядра.

*Определение 2.2.6*

**Многопоточность** (англ. Multithreading — SMT) — свойство платформы (например, операционной системы, виртуальной машины и т.д.) или приложения, состоящее в том, что процесс, порожденный в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

*Определение 2.2.7*

**Одновременная многопоточность** (англ. Simultaneous Multithreading — SMT) — одна из двух главных форм многопоточности, которая может быть реализована в процессорах аппаратно. Второй формой является временная многопоточность. Технология одновременной многопоточности позволяет исполнять инструкции из нескольких независимых потоков выполнения на множестве функциональных модулей суперскалярного микропроцессора в одном цикле.

*Определение 2.2.8*

**Временная многопоточность** (англ. Temporal multithreading). Различие между этими двумя формами состоит в максимальном количестве потоков, которые исполняются на каждой стадии вычислительного конвейера в определенный тактовый цикл процессора. При временной многопоточности в каждый данный момент исполняется только один поток, а при одновременной многопоточности — несколько. Некоторые специалисты используют термин super-threading в качестве синонима временной многопоточности.

На обычном процессоре управление потоками осуществляется операционной системой. Поток исполняется до тех пор, пока не произойдет аппаратное прерывание, системный вызов или пока не истечет отведенное для него операционной системой время. После этого процессор переключается на код операционной системы, который сохраняет состояние потока (его контекст) и переключается на состояние следующего в очереди потока, которому тоже выделяется время на исполнение. При такой многопоточности достаточно большое количество тактов процессора тратится на код операционной системы, переключающий контексты. Если поддержку потоков реализовать аппаратно, то процессор сам сможет переключаться между потоками, а в идеальном случае — выполнять несколько потоков одновременно за каждый такт.

Одновременная многопоточность объединяет параллельное исполнение инструкций суперскалярной архитектуры с аппаратной многопоточностью. Применение одновременной многопоточности, благодаря динамическому распределению функциональных модулей процессора между потоками, увеличивает использование процессора при наличии задержек памяти и ограниченной возможности параллельного исполнения инструкций внутри потока.

Многоядерные микропроцессоры по своей организации наиболее близки к микропроцессорам с одновременной многопоточностью — имеют несколько наборов регистров, несколько функциональных модулей и суперскалярность каждого из ядер. Главное отличие между ними заключается в распределении ресурсов — в многоядерном процессоре каждый поток получает фиксированное количество функциональных модулей процессора, тогда как в процессоре с одновременной многопоточностью распределение модулей изменяется в каждом цикле. Вследствие этого процессоры с одновременной многопоточностью показывают большую производительность при максимальной загрузке потоками, а при снижении количества потоков производительность падает медленней, по сравнению с многоядерным процессором.

*Определение 2.2.9*

**Гиперпоточность** (англ. hyper-threading technology, HTT или HT) — технология, разработанная компанией Intel, позволяющая ядру процессора исполнять больше потоков данных чем один (обычно два). Так как было выяснено, что обычный процессор в большинстве задач использует не более 70% всей вычислительной мощности, было решено использовать технологию, позволяющую при простое определенных вычислительных блоков — нагрузить их работой с другим потоком. Это позволяет увеличить производительность ядра от 10 до 80% в зависимости от задачи.

**Кэш-память**: во всех существующих на сегодня многоядерных процессорах кэш-памятью 1-го уровня обладает каждое ядро в отдельности, а кэш-память 2-го уровня существует в нескольких вариантах:

* разделяемая — расположена на одном кристалле с ядрами и доступна каждому из них в полном объеме.
* индивидуальная — отдельные кэши равного объема, интегрированные в каждое из ядер.

Многоядерные процессоры также имеют гомогенную или гетерогенную архитектуру:

* гомогенная архитектура — все ядра процессора одинаковы и выполняют одни и те же задачи
* гетерогенная архитектура — ядра процессора выполняют разные задачи.

Подавляющее большинство современных процессоров имеют два и более ядра. **Топовые** модели могут содержать и 8, и даже 12 ядер. Мы практически получаем несколько процессоров, способных независимо решать каждый свои задачи, при этом, естественно, возрастает производительность. Однако прирост производительности далеко не всегда оправдывает ожидания.

Во-первых, далеко не все программы поддерживают распределение вычислений на несколько ядер. Естественно, можно программы разделять между ядрами, чтобы на каждом ядре работал свой набор независимых программ. Например, на одном ядре работает операционная система с набором служебных программ, на другом пользовательские программы и так далее.

Но это дает выигрыш в производительности до тех пор, пока не появляется программа, требующая ресурсов больше, чем может дать одно ядро. Хорошо, если она поддерживает распределение нагрузки между несколькими ядрами

Во-вторых, усложняется работа с памятью, так как ядер – много, и всем им требуется доступ к ОЗУ. Требуется сложный механизм, определяющий очередность доступа ядер процессора к памяти и к другим ресурсам компьютера.

В-третьих, возрастает энергопотребление, а, следовательно, увеличивается тепловыделение и требуется мощная система охлаждения.

**Синхронная и асинхронная работа ядер**

Синхронная загрузка означает, что совершенно одинаковые операции осуществляются независимо над разными кусочками данных. Эту обработку можно осуществлять последовательно на одном ядре. Бывает выгодно дать разные кусочки всем ядрам для обработки. Все ядра, т.к. у каждого, например, свое собственное независимое АЛУ, если начали обработку разных кусочков по одинаковой операции в один момент времени, то синхронно закончат так же в один момент времени.

Если надо выполнить много разных операций, то синхронной работы не выйдет, т.к. выполняя разные операции с разными данными, ядра заканчивают в разный момент, т.к. для каждой команды потратится разное время.

Синхронный режим — это когда процессор и память работают на одной частоте шины.

Асинхронный режим — когда шины процессора и памяти находятся в соотношении 2:3, 4:5 и т.д.

В многоядерных процессорах появились два новых элемента: синхронизатор и арбитр.

*Определение 2.2.10*

**Арбитр** - специальное устройство, позволяющее обеспечить доступ с низкой латентностью к интерфейсу FSB.

*Определение 2.2.11*

**Front Side Bus** (FSB, системная шина) — шина, обеспечивающая соединение между x86/x86-64-совместимым центральным процессором и внутренними устройствами.

На каждом ядре существует специальный служебный элемент, посредством которого арбитр связан с самим ядром - синхронизатор. Арбитр работает на фиксированной частоте, кратной частоте FSB.

Асинхронный интерфейс между арбитром и каждым ядром позволяет изменять частоту работы ядра и кэша по необходимости. Разумеется, наличие арбитра обуславливает появление соответствующей латентности при общении ядра и системного интерфейса. При прочих равных условиях приоритет отдается запросам на чтение, как наиболее важным.

*Определение 2.2.12*

**Application-specific instruction-set processor** **(ASIP)** (рус. «проблемно-ориентированный процессор», «заказной микропроцессор», «заказная микросхема») — компонент (как правило, процессорное ядро), используемый в проектировании [систем на кристалле](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BD%D0%B0_%D0%BA%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%BB%D0%BB%D0%B5).

*Определение 2.2.13*

**Система на кристалле** или **однокристальная система** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *System-on-a-Chip*) — в [микроэлектронике](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0) — электронная схема, выполняющая функции целого устройства (например, [компьютера](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80)) и размещенная на одной [интегральной схеме](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D1%85%D0%B5%D0%BC%D0%B0).

[Система команд](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4) ASIP специально спроектирована для выполнения специфичных программ. Подобная специализация ядра позволяет достичь компромисса между универсальностью [процессора](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) общего назначения (CPU) и производительностью [ASIC](https://ru.wikipedia.org/wiki/ASIC).

*Определение 2.2.14*

**«Жесткий» процессор** — процессор, в котором набор инструкций задается жестко, каждая машинная инструкция (сложение, сдвиг, копирование) воплощена непосредственно в схеме. Чем больше инструкций, тем сложнее схема.

В процессорах, использующих микрокод, на схеме реализованы только основные команды, остальные инструкции реализованы в микропрограммах. Это позволяет не тратится на создание сложных схем, реализуя большое множество инструкций процессора. Подход, при котором используются сложные микрокодовые инструкции, называют CISC.

2.3. Представление о микросхемах обвязки процессоров

*Определение 2.3.1*

**Чипсет** (англ. chipset) - набор микросхем обычно применяемых, при создании системных плат, для "обвязки" процессора и спроектированных для совместной работы с целью выполнения набора каких-либо функций.

Так, в компьютерах чипсет, размещаемый на материнской плате, выполняет роль связующего компонента, обеспечивающего совместное функционирование подсистем памяти, центрального процессора, ввода-вывода и других. Чипсеты встречаются и в других устройствах, например, в сотовых телефонах.

*Определение 2.3.2*

**Обвязка** - вспомогательные части системы, обеспечивающие работу основного элемента, и как правило расположенные вокруг него.

Микросхемы обвязки процессоров — вспомогательные микросхемы, расположенные вокруг процессора.

В современных процессорах внутри корпуса уже расположены несколько шин, видео карта, и несколько контроллеров шин. Постепенно так получается, что обвязки для современных процессоров требуется все меньше и меньше. Но это не значит, что ее нет, процессору по-прежнему нужны шины, нужны внешние устройства, но их постепенно удается уместить в корпус процессора. При этом надо понимать, что с точки зрения архитектуры, хотя все это и находится в одном корпусе, все же это отдельные части, которые просто находятся под одной крышкой.

*Определение 2.3.3*

**Компьютерная шина** – подсистема, осуществляющая передачу между функциональными блоками компьютера.

Компьютеры имеют как правило внутренние и внешние шины.  
Внутренние шины необходимы для подключения компонентов, расположенных внутри системного блока. К важнейшим внутренним шинам относятся:

* системная шина с разъемом процессора;
* шина памяти с разъемами модулей памяти;
* шина и слот видеокарты;
* шины и слоты плат расширения;
* шины и порты накопителей;
* шина и разъемы электропитания;
* линии и порты интерфейса управления питанием;
* порты и панели индикации;
* шины и порты управления системой.

Внешние шины необходимыдля подключения компонентов и периферии, расположенных вне системного блока. Среди внешних интерфейсов следует особо выделить группу, обслуживающую обязательные компоненты компьютерной системы: монитор, клавиатуру, мышь. Прочие внешние интерфейсы обслуживают дополнительные внешние устройства, объединяемые понятием «периферия». К типовым внешним интерфейсам относятся:

* *порты* видеокарты для подключения мониторов и телевизора;
* порты PS/2 для клавиатуры и мыши;
* шина и порты USB;
* последовательные порты СОМ;
* параллельные порты LPT;
* последовательный порт FireWire;
* сетевой порт RJ45;
* порт модема RJ11 для телефонной линии;
* порты аудиоинтерфейса.

*Определение 2.3.4*

**Аппаратный порт** - специализированный разъем в компьютере, предназначенный для подключения оборудования определенного типа.

Обычно портами называют разъемы, предназначенные для работы периферийного оборудования, существенно разделенного от архитектуры компьютера.

**Северный и южный мосты компьютера** (а правильнее будет сказать, материнской платы) - это два основных функциональных контроллера, которые отвечают за работу всех компонентов системной платы и называются **чипсетом** (англ. chipset).

*Определение 2.3.5*

**Северный мост** (англ. Northbridge) — это системный контроллер, являющийся одним из элементов чипсета материнской платы, отвечающий за работу с оперативной памятью (RAM), видеоадаптером и процессором (CPU).

Северный мост отвечает за частоту системной шины, тип оперативной памяти и ее максимально возможный объем. Одной из основных функций северного моста является обеспечение взаимодействия системной платы и процессора, а также определение скорости работы. Частью северного моста во многих современных материнских платах является встроенный видеоадаптер. Таким образом, функциональная особенность северного моста являет собой еще и управление шиной видеоадаптера и ее быстродействием. Также северный мост обеспечивает связь всех вышеперечисленных устройств с южным мостом.

*Определение 2.3.6*

**Южный мост** (англ. Southbridge) — это функциональный контроллер, известен как контроллер ввода-вывода или ICH (In/Out Controller Hub). Отвечает за так называемые "медленные" операции, к которым относится отработка взаимодействия между интерфейсами IDE, SATA, USB, LAN, Embeded Audio и северным мостом системы, который, в свою очередь, напрямую связан с процессором и другими важными компонентами, такими как оперативная память или видеоподсистема. Также южный мост отвечает за обработку данных на шинах PCI, PCIe.

*Определение 2.3.7*

**Таймер** - многофункциональное времязадающее устройство, представляющие ступень между аналоговыми и цифровыми устройствами. Он позволяет производить преобразование аналоговых сигналов в последовательность импульсов по заданному закону.

*Определение 2.3.9*

**Контроллер прерываний** (англ. Programmable Interrupt Controller, PIC) - микросхема или встроенный блок процессора, отвечающий за возможность последовательной обработки запросов на прерывание от разных устройств.

*Определение 2.3.10*

**Цифро-аналоговый преобразователь (ЦАП)** - устройство для преобразования цифрового (обычно двоичного) кода в аналоговый сигнал (ток, напряжение или заряд). Цифро-аналоговые преобразователи являются интерфейсом между дискретным цифровым миром и аналоговыми сигналами.

2.4. Представление о конвейерном выполнении команд в процессорах

Выполнение каждой команды складывается из ряда последовательных этапов (шагов, стадий), суть которых не меняется от команды к команде. С целью увеличения быстродействия процессора и максимального использования всех его возможностей в современных микропроцессорах используется конвейерный принцип обработки информации. Этот принцип подразумевает, что в каждый момент времени процессор работает над различными стадиями выполнения нескольких команд, причем на выполнение каждой стадии выделяются отдельные аппаратные ресурсы. По очередному тактовому импульсу каждая команда в конвейере продвигается на следующую стадию обработки, выполненная команда покидает конвейер, а новая поступает в него.

На рисунке 2.4.1 изображен конвейер из 5 блоков, которые называются ступенями. Первая ступень (блок С1) вызывает команду из памяти и помещает ее в буфер, где она хранится до тех пор, пока не потребуется. Вторая ступень (блок С2) декодирует эту команду, определяя ее тип и тип ее операндов. Третья ступень (блок С3) определяет местонахождение операндов и вызывает их из регистров или из памяти. Четвертая ступень (блок С4) выполняет команду, обычно проводя операнды через тракт данных. И наконец, блок С5 записывает результат обратно в нужный регистр.

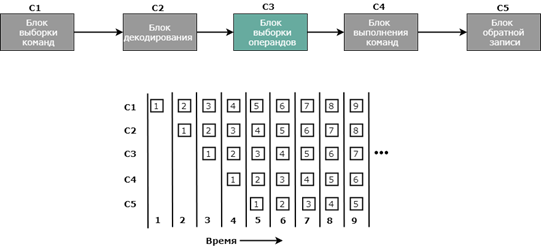


Рис. 2.4.1 – Пятиступенчатый конвейер. Состояние каждой ступени в зависимости от кол-ва пройденных циклов

1 цикл: блок С1 обрабатывает команду 1, вызывая ее из памяти.

2 цикл: блок С2 декодирует команду 1, в то время как блок С1 вызывает из памяти команду 2.

3 цикл: блок С3 вызывает операнды для команды 1, блок С2 декодирует команду 2, а блок С1 вызывает команду 3.

4 цикл: блок С4 выполняет команду 1, С3 вызывает операнды для команды 2, С2 декодирует

команду 3, а С1 вызывает команду 4.

5 цикл: блок С5 записывает результат выполнения команды 1 обратно в регистр, тогда как другие ступени конвейера обрабатывают следующие команды.

**Наборы команд и взаимодействие с операндами с позиций длины и операндов**

1. **Процессор с сокращенным набором команд**

*Определение 2.4.1*

**RISC** (англ. restricted (reduced) instruction set computer — «компьютер с сокращенным набором команд») — архитектура процессора, в котором быстродействие увеличивается за счет упрощения команд, чтобы их декодирование было более простым, а время выполнения — меньшим.

Недостатки RISC прямо связаны с некоторыми преимуществами этой архитектуры. Принципиальный недостаток — сокращенное число команд: на выполнение ряда функций приходится тратить несколько команд вместо одной в CISC. Это удлиняет код программы, увеличивает загрузку памяти и трафик команд между памятью и ЦП. Исследования показали, что RISC-программа в среднем на 30% длиннее CISC-программы, реализующей те же функции.

Хотя большое число регистров дает существенные преимущества, само по себе оно усложняет схему декодирования номера регистра, тем самым увеличивается время доступа к регистрам.

Устройство управления с аппаратной логикой, реализованное в большинстве RISC-систем, менее гибко, более склонно к ошибкам, затрудняет поиск и исправление ошибок, уступает при выполнении сложных команд.

1. **Процессор с расширенным набором команд**

*Определение 2.4.2*

**CISC** (  Complex Instruction Set Computing - компьютер с полным набором команд) это архитектура процессора где одиночные команды могут выполнять несколько операций низкого уровня (например, загрузка из памяти, арифметические операции, хранилище памяти) или способны на многоступенчатые операции или режимы адресации в пределах одной инструкции.

Основную идею CISC-архитектуры отражает ее название - «полный набор команд». В данной архитектуре стремятся иметь отдельную машинную команду для каждого возможного (типового) действия по обработке данных.  
Исторически CISC-архитектура была одной из первых. Совершенствование процессоров шло по пути создания ВМ, способных выполнять как можно больше разных команд. Это упрощало работу программистов, которые писали программы на языке ассемблера (то есть практически на уровне машинных команд). Использование сложных команд позволяло сократить размер и время разработки программы.  
  
В итоге сложились следующие черты организации CISC-процессоров:

* большое количество различных машинных команд (сотни), каждая из которых выполняется за несколько тактов центрального процессора;
* устройство управления с программируемой логикой;
* небольшое количество регистров общего назначения;
* различные форматы команд с разной длиной;
* преобладание двухадресной адресации;
* развитый механизм адресации операндов, включающий различные методы косвенной адресации.

CISC-подход, однако, привел к тому, что некоторые команды стало невозможно выполнять чисто аппаратными средствами (при разумной сложности таких средств). В результате в процессорах появились блоки, «на лету» заменяющие наиболее сложные команды последовательностями из более простых команд. Мало того, практика показала, что многие сложные команды при написании программ оказывались просто не востребованы. Наконец, из-за высокой сложности команд и их обилия устройство управления ВМ приходилось строить только на основе программируемой логики, то есть с применением «медленной» управляющей памяти. Последнее обстоятельство существенно ограничивало возможности наращивания тактовой частоты процессора. Все эти факторы привели к повороту в сторону RISC-архитектуры. В то же время целый ряд несомненных достоинств CISC-архитектуры сохраняют ее актуальность (прежде всего, в глазах разработчиков программных приложений). Именно поэтому ведущие фирмы-производители ВМ (Intel, AMD, IBM и др.) в своих последних разработках, по-прежнему, не отказываются от CISC-подхода.

1. **Процессор со сверхдлинными командами**

*Определение 2.4.3*

**VLIW** (англ. very long instruction word — «очень длинная машинная команда») – архитектура процессоров, характеризующаяся возможностью объединения нескольких простых команд в так называемую связку. Входящие в нее команды должны быть независимы друг от друга и выполняться параллельно. Таким образом, из нескольких независимых машинных команд транслятор формирует одно «очень длинное командное слово».

Идея VLIW базируется на том, что задача эффективного планирования параллельного выполнения команд возлагается на «разумный» компилятор. Такой компилятор вначале анализирует исходную программу. Цель анализа: обнаружить все команды, которые могут быть выполнены одновременно, причем так, чтобы между командами не возникали конфликты. В ходе анализа компилятор может даже частично имитировать выполнение рассматриваемой программы. На следующем этапе компилятор пытается объединить такие команды в пакеты (связки), каждый из которых рассматривается как одна сверхдлинная команда. Объединение нескольких простых команд в одну сверхдлинную производится по следующим правилам:

* количество простых команд, объединяемых в одну команду сверхбольшой длины, равно числу имеющихся в процессоре функциональных (исполнительных) блоков (ФБ);
* в сверхдлинную команду входят только такие простые команды, которые исполняются разными ФБ, то есть обеспечивается одновременное исполнение всех составляющих сверхдлинной команды.

Длина сверхдлинной команды обычно составляет от 256 до 1024 битов. Такая метакоманда содержит несколько полей (по числу образующих ее простых команд), каждое из которых описывает операцию для конкретного функционального блока.

В качестве простых команд, образующих сверхдлинную, обычно используются команды RISC-типа.

То, что в выполняемой сверхдлинной команде исключена возможность конфликтов, позволяет предельно упростить аппаратуру VLIW-процессора и, как следствие, добиться более высокого быстродействия. Подавляющее большинство цифровых сигнальных процессоров и мультимедийных процессоров с производительностью более 1 млрд операций/с базируется на VLIW-архитектуре.

Какие же есть недостатки у VLIW. Код для VLIW обладает невысокой плотностью. Из-за большого количества пустых инструкций для простаивающих устройств программы для VLIW-процессоров могут быть гораздо длиннее, чем аналогичные программы для традиционных архитектур.

Из-за сложных внутренних зависимостей кода, программирование на уровне машинных кодов для VLIW-архитектур практически невозможно вручную. Приходится полагаться на оптимизацию компилятора, который сам может содержать ошибки.

Проблемами VLIW-архитектуры являются:

1) усложнение регистрового файла и, прежде всего, связей этого файла с вычислительными устройствами;

2) трудности создания компиляторов, способных найти в программе независимые команды, связать такие команды в длинные строки и обеспечить их параллельное выполнение.

**Наборы команд и взаимодействие с операндами с позиций количества независимых потоков данных**

*Определение 2.4.4*

**SISD** (Single Instruction stream over a Single Data stream) - вычислительная система с одиночным потоком команд и одиночным потоком данных.

**Архитектура SISD** — это компьютер архитектуры фон-Неймана с одним процессором, который, работая с одним потоком данных, выполняет последовательно инструкции. SISD-машина не является параллельной, т.к. в данном классе не используется параллелизм данных и инструкций. К этому классу также принято относить суперскалярные, конвейерные и VLIW-процессоры.

**SIMD**

В SIMD-компьютеры входит один командный процессор (управляющего модуля), который называется контроллером, и несколько модулей обработки данных - процессорные элементы. Управляющий модуль принимает, анализирует и выполняет команды. Если в команде встречаются данные, контроллер рассылает на все процессорные элементы команду, которая выполняется на некотором числе процессорных элементов, каждый из которых имеет свою собственную память для хранения данных. Преимущество данной архитектуры в том, что в этом случае логика вычислений реализована более эффективно. Почти половина логических инструкций обычного процессора связана с управлением выполнением машинных команд, а остальная часть относится к выполнению арифметических операций и работе с внутренней памятью процессора.

**Архитектура ILP- процессоров**

Более высокая производительность достигается за счет совершенствования полупроводниковой технологии и за счет увеличения плотности микросхем. С помощью реализации определенного вида параллелизма можно также добиться увеличения скорости выполнения программ. Параллелизм на уровне команд (instruction-level parallelism, ILP) стал возможен благодаря созданию методик компиляции и процессоров, которые за счет параллельного выполнения отдельных RISC-операций ускоряют работу. Системы на базе ILP используют программы, написанные на традиционных языках высокого уровня для последовательных процессоров, а обнаружение «скрытого параллелизма» автоматически выполняется с помощью применения соответствующей компиляторной технологии и аппаратного обеспечения.

**Суперскалярные процессоры**

*Определение 2.4.5*

**Суперскалярные процессоры** – это реализация ILP-процессора для последовательных архитектур, программа для которых не должна и не может передавать точную информацию о параллелизме. Т.к. программа не содержит точной информации о наличии ILP, то задача обнаружения параллелизма должна решаться аппаратурой, которая в свою очередь должна создавать план действий для обнаружения «скрытого параллелизма». Код для суперскалярных процессоров содержит последовательность команд, порождающую корректный результат, если выполняется в установленном порядке. Код указывает последовательный алгоритм и не представляет себе точную природу аппаратного обеспечения, на котором он будет работать или точный временной порядок, в котором будут выполняться команды (за исключением того, что он использует конкретный набор команд). Такой подход увеличивает сложность аппаратного обеспечения. Кроме того, суперскалярный процессор создает план выполнения, при этом используя преимущества тех факторов, которые могут быть определены только во время выполнения.

Допустимые границы преобразования динамической структуры программы задают существующие на множестве инструкций отношения: зависимость по данным и по управлению. При описании архитектур суперскалярных процессоров обычно используется модель окна исполнения. При исполнении программы микропроцессор как бы продвигает по статической структуре программы окно исполнения. Если между командами нет зависимости, они могут исполняться в окне параллельно.

*Определение 2.4.6*

**Тип архитектуры EPIC** ( EPIC - Explicitly Parallel Instruction Computing – “вычисление с явным параллелизмом машинных команд”) – это эволюция архитектуры VLIW, которая абсорбировала в себе многие концепции суперскалярной архитектуры, в форме, адаптированной к EPIC. EPIC – это «идеология», определяющая, как создавать ILP-процессоры, а также набор характеристик архитектуры, которые поддерживают данную идеологию. К архитектуре EPIC можно отнести множество разных архитектур набор команд (ISA). Помимо включения или исключения различных характеристик архитектуры, проектировщики должны принимать решения по набору кодов операций, диапазону поддерживаемых типов данных и количеству используемых регистров. Любая архитектура, относящаяся к классу EPIC, должна реализовывать общую идеологию EPIC. Она может быть оптимизирована для различных приложений (например, для систем общего назначения или встроенных устройств), в зависимости от того, какие из характеристик EPIC использует архитектура ISA. Поэтому тип архитектуры EPIC должен быть архитектурой общего назначения, способной достигать высокой степени параллелизма на уровне команд в числовых и скалярных приложениях.

Одна из целей создания архитектуры EPIC - сохранить реализованный в VLIW принцип статического создания плана выполнения, при этом обогатить его возможностями суперскалярного процессора, позволившими новой архитектуре лучше учитывать динамические факторы, традиционно ограничивающие параллелизм. Таким образом, «идеология» EPIC была построена на следующих основных принципах:

* Компилятор должен играть ключевую роль в создании плана выполнения, а архитектура должна обеспечивать поддержку компонентов, необходимых для этого.
* Архитектура должна обеспечивать функции, помогающие компилятору в использовании статических ILP.
* Архитектура должна обеспечивать механизм для передачи плана выполнения компилятора аппаратному обеспечению.

2.5. Классификация и особенности шин в отношении способа передачи, метода синхронизации, топологии, способа управления, адресации устройств

*Определение 2.5.1*

**Цифровой последовательной передачей** называется последовательная отправка [битов](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) по одному проводу, оптическому пути или частоте. В связи с тем, что это требует меньшей [обработки сигнала](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB%D0%BE%D0%B2) и меньше вероятность ошибки, чем при параллельной передаче, то скорость передачи данных по каждому отдельному пути может быть быстрее. Этот механизм может использоваться на более дальних расстояниях, потому что легко может быть передана контрольная цифра или бит четности.

*Определение 2.5.2*

**Параллельная передача** — это такая одновременная передача соответствующих элементов сигнала по двум или большему числу путей. Используя множество электрических проводов можно передавать несколько бит одновременно, что позволит достичь более высоких скоростей передачи, чем при последовательной передаче. Этот метод применяется внутри компьютера (например, во внутренних [шинах данных](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D0%BD%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)), а иногда и во внешних устройствах(например, [принтеры](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%82%D0%B5%D1%80)). Основной проблемой при этом является «перекос», потому что провода при параллельной передаче имеют немного разные свойства, следовательно, некоторые биты могут прибыть раньше других, а это может повредить сообщение. Бит четности может способствовать сокращению ошибок. Тем не менее, электрический провод при параллельной передаче данных менее надежен на больших расстояниях, так как передача нарушается с очень большой вероятностью.

Главным отличием последовательных шин от параллельных является сам способ передачи данных.

**Последовательные** шинах используют одну сигнальную линию (возможно использование двух отдельных каналов для разделения потоков приема-передачи). А значит, что информационные биты здесь передаются последовательно. Данные для передачи через последовательную шину облекаются в пакеты (единица информации, передаваемая как целое между двумя устройствами), которые, сверх собственных полезных данных, включают еще некоторое количество служебной информации.

В **параллельных шинах** рассматривается совокупность сигнальных линий (проводников), объединенных по их назначению (управление, адреса, данные), которые имеют определенные электрические характеристики и протоколы передачи информации. Группы этих сигнальных линий также называются шинами:

* Линии для управления данными (шина управления);
* Линии для адресации данных (шина адреса);
* Линии для обмена данными (шина данных);

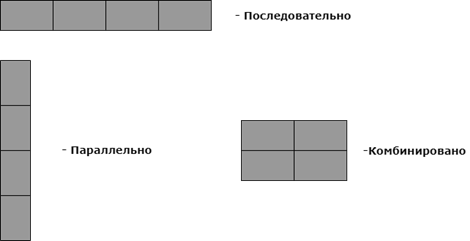
**Комбинированная шина:**

В каких-то случаях выгодно огромный кусок данных разбить так, чтобы некоторые его части передавалась последовательно, а какие-то - параллельно.

Например, нужно передать 4 элемента данных.

Возможные варианты передачи:

1. Комбинированно: элементы данных объединяют по два и используют 2 такта для передачи.
2. Передать за 4 такта времени по последовательной шине.
3. Передать за 1 такт времени по параллельной шине.

 Рис. 2.5.1. Варианты передачи 4 элементов данных

**Классификация и особенности шин в отношении метода синхронизации:**

(Синхронизируются 2 объекта: передатчик и приемник. Синхронизация — передача данных)

*Определение 2.5.3*

Для последовательной передачи данных достаточно одной линии, по которой могут последовательно передаваться биты данных. Приемник обязан уметь распознавать, где начинается и где заканчивается сигнал, который отвечает каждому биту данных, то есть передатчику и приемнику нужно уметь синхронизироваться. В случае, если качество синхронизации низкое (за время передачи одного бита несогласованность достигает нескольких процентов), тогда используется **асинхронный** режим передачи данных: выполняется согласование синхрогенераторов (генерирует импульс, называемый импульсом синхронизации) в начале передачи каждого байта.

Как правило, передача байта начинается из специального старт-бита, потом идут биты данные, а за ними, возможно, бит четности (Рис. 2.6.2). После всех битов данных передается стоп-бит. Старт-бит и стоп-бит всегда имеют определенное значение: старт-бит кодируется логическим нулем, а стоп-бит - логической единицей. Между передачей стоп-бита одного байта и старт-бита следующего байта может проходить произвольное время.

Асинхронный режим очень зависит от погрешностей синхрогенераторов, что задает моменты приема битов. Чем выше скорость передачи, тем выше эта погрешность. В результате всех ограничений скорость передачи в асинхронном режиме ограничена сотнями килобит в секунду (стандартные скорости: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 бит/с).

*Определение 2.5.4*

Если синхронизация весьма качественна (к примеру, используется дополнительная линия, по которой передается синхросигнал), то можно передавать поток данных без дополнительной синхронизации отдельных байтов. Такой режим называется **синхронным**. Передача битов данных опережается и заканчивается выдачей в канал символа синхронизации. Если данные отсутствуют, то передатчик должен постоянно передавать в канал символы синхронизации.

Синхронная последовательная передача начинается с пересылки в приемник одного или двух символов синхронизации (нельзя путать с импульсами синхронизации). Приемник, получив такой символ(ы), начинает прием данных и их преобразование в параллельный формат. Очевидно, что при такой организации синхронной последовательной передачи она целесообразна лишь для пересылки массивов слов, но не отдельных символов. Это помешало широкому распространению синхронной последовательности передачи данных.

*Определение 2.5.5*

**Пакетный** режим передачи данных позволяет ускорить процессы чтения данных и их запись в память. Если требуется чтение из памяти (записи в память) одного машинного слова, процессор считывает вместе с ним еще несколько подряд расположенных слов. Длина каждого машинного слова (размер отдельного блока данных, передаваемого между микропроцессором и памятью), равна разрядности внешней шины данных микросхемы памяти. Ширина шины данных самых первых микросхем памяти равнялась всего 1 биту, сейчас чаще можно встречают 4-, 8- и 16- (реже 32-) битные микросхемы памяти.

При пакетной передаче данных нет необходимости указывать полный адрес (номер строки и столбца) каждой ячейки памяти. Вместо этого подается только адрес начальной ячейки пакета, а следующие ячейки считываются подряд из текущей строки столько раз, сколько слов в пакете. Преимущество такой схемы в экономии времени на передачу адреса: для чтения нескольких слов данных требуется указать всего лишь один адрес.

В большинстве архитектур SDRAM подсистема памяти высылает критическое слово в первую очередь, а после идут остальные данные. Так работает память SDRAM в пакетном режиме (burst mode). Память высылает на шину четыре блока по 8 байт (или один пакет). Там же присутствует задержка в три тактовых импульса между отсылкой запроса и поступлением на шину первого из четырех слов, если задержка чтения в нашей системе составляет 3 такта шины памяти. Поэтому первое критическое слово имеет задержку в 3 такта, и каждый последующий 8-байтный блок — задержку в один такт. Это соответствует задержкам DRAM пакетного режима типа 3-1-1-1. Общее время, требуемое на передачу всех данных по шине, составляет 3+1+1+1=6 тактов шины памяти. Если бы пакетного режима не существовало, тогда на передачу того же количества данных ушло бы 3+3+3+3=12 тактов шины памяти.

**Классификация и особенности шин в отношении топологии**

Топология определяет способ взаимодействия компьютеров в сети. Различным видам топологий соответствуют различные методы взаимодействия, и эти методы оказывают сильное влияние на сеть. Все сети строятся на основе трех базовых топологий: *шина (bus); звезда (star); кольцо (ring)*. Если компьютеры подключены вдоль одного кабеля (сегмента), топология называется шиной. В том случае, когда компьютеры подключены к сегментам кабеля, исходящим из одной точки, или концентратора, топология называется звездой. Если кабель, к которому подключены компьютеры, замкнут в кольцо, такая топология носит название кольца. Хотя сами по себе базовые топологии несложны, в реальности часто встречаются довольно сложные комбинации, объединяющие свойства нескольких топологий.

*Определение 2.5.6*

Топологию **"шина"** зачастую называют "линейной шиной" (linear bus). Эта топология является самой простой и широко распространенной топологией. В ней используется один кабель, который называется магистралью или сегментом, вдоль которого параллельно подключены все компьютеры сети. В этой сети компьютеры адресуют данные конкретному компьютеру, передавая их по кабелю в виде электрических сигналов. Данные в виде электрических сигналов передаются всем компьютерам сети, однако информацию принимает только тот, адрес которого соответствует адресу получателя, зашифрованному в этих сигналах. Причем в каждый момент времени только один компьютер может вести передачу. Из-за того, что данные в сеть передаются лишь одним компьютером, то ее производительность зависит от количества компьютеров, подключенных к шине. Чем больше компьютеров, ожидающих передачи данных, тем медленнее сеть. (На быстродействие сети влияет множество факторов: частота, с которой компьютеры передают данные; тип сетевого кабеля; характеристики аппаратного обеспечения компьютеров в сети; тип работающих сетевых приложений; расстояние между компьютерами в сети.) Шина является пассивной топологией. Это означает, что компьютеры только "слушают" передаваемые по сети данные, но не перемещают их от отправителя к получателю. Поэтому, если один из компьютеров выйдет из строя, то это не скажется на работе остальных. В активных топологиях компьютеры регенерируют сигналы и передают их по сети. Данные, или электрические сигналы, распространяются по всей сети (от одного конца кабеля к другому). Если не предпринимать никаких специальных действий, тогда сигнал, достигая конца кабеля, будет отражаться и не позволит другим компьютерам осуществлять передачу. В связи с этим, после того, как данные достигнут адресата, электрические сигналы необходимо погасить. Чтобы предотвратить отражение электрических сигналов, на каждом конце кабеля устанавливают терминаторы, которые поглощают эти сигналы. Все концы сетевого кабеля должны быть к чему-нибудь подключены (например, к компьютеру) для увеличения длины кабеля.

*Определение 2.5.7*

При топологии **"звезда"** все компьютеры с помощью сегментов кабеля подключаются к центральному компоненту, именуемому концентратором (hub). Сигналы от передающего компьютера поступают через концентратор ко всем остальным. В сетях с топологией "звезда" подключение кабеля и управление конфигурацией сети централизованы. Но есть и минус: так как все компьютеры подключены к центральной точке, для больших сетей значительно увеличивается расход кабеля. Если центральный компонент выйдет из строя, то нарушится работа всей сети. А если выйдет из строя только один компьютер (или кабель, соединяющий его с концентратором), тогда лишь этот компьютер не сможет передавать или принимать данные по сети, при этом на остальные компьютеры в сети это не повлияет.

*Определение 2.5.8*

При топологии **"кольцо"** компьютеры подключаются к кабелю, который замкнут в кольцо. Очевидно, что у этого кабеля просто не может быть свободного конца, к которому надо подключать терминатор. Сигналы передаются по кольцу в одном направлении и проходят через каждый компьютер. В отличие от пассивной топологии "шина", здесь каждый компьютер выступает в роли репитера, усиливая сигналы и передавая их следующему компьютеру. Поэтому, если выйдет из строя один компьютер, престает функционировать вся сеть. Один из принципов передачи данных в кольцевой сети носит название передачи маркера. Суть его такова. Маркер последовательно (т.е. от одного компьютера к другому) передается до тех пор, пока его не получит тот, который "хочет" передать данные. Передающий компьютер изменяет маркер, помещает электронный адрес в данные и посылает их по кольцу. Данные проходят через каждый компьютер, пока не окажутся у того, чей адрес совпадает с адресом получателя, указанным в данных. После чего принимающий компьютер посылает передающему компьютеру сообщение, где подтверждает факт приема данных. Передающий компьютер, получив подтверждение, создает новый маркер и возвращает его в сеть.

*Определение 2.5.9*

**Комбинированными топологиями**называются такиетопологии, которые комбинируют компоновку сети по принципу шины, звезды и кольца. *Звезда-шина (star-bus)* — это комбинация топологий "звезда" и "шина". Если есколько сетей с топологией "звезда" объединяются при помощи магистральной линейной шины, тогда выход из строя одного компьютера не оказывает никакого влияния на сеть - остальные компьютеры по-прежнему взаимодействуют друг с другом. А выход из строя концентратора повлечет за собой остановку подключенных к нему компьютеров и концентраторов.

*Определение 2.5.10*

**Звезда-кольцо** *(star-ring)* можно сказать, немного похожа на звезду-шину. В них обеих топологии компьютеры подключены к концентратору, который формирует шину или кольцо. Различие в том, что концентраторы в звезде-шине соединены магистральной линейной шиной, а в звезде-кольце на основе главного концентратора они образуют звезду.

Таблица. 2.6.3

|  |  |  |
| --- | --- | --- |
| Топология | Преимущества | Недостатки |
| Шина | Экономный расход кабеля. Сравнительно недорогая и сложная в использовании среда передачи. Простота, надежность. Легко расширяется | При значительных объемах графика уменьшается пропускная способность среды. Трудно локализовать проблемы. Выход из строя кабеля останавливает работу многих пользователей |
| Кольцо | Все компьютеры имеют равный доступ. Количество пользователей не оказывает сколько-нибудь значительного влияния на производительность | Выход из строя одного компьютера может вывести из строя всю сеть. Трудно локализовать проблемы. Изменение конфигурации сети требует остановки работы всей сети |
| Звезда | Легко модифицировать сеть, добавляя новые компьютеры. Централизованный контроль и управление. Выход из строя одного компьютера не влияет на работоспособность сети. | Выход из строя центрального узла выводит из строя всю сеть. |

**Классификация и особенности шин в отношении адресации устройств (индивидуальная, групповая, широковещательная).**

Разделение шин в отношении адресации устройств зависит от назначения шины. Индивидуальная адресация применяется для простой связи двух устройств типа “точка-точка”. Но чаще всего требуется связать большое число устройств (устройства соединены шиной и у каждого устройства есть адрес), для этого и нужны групповая и широковещательная адресации.

Широковещательная адресация, подразумевает, что есть сетевая станция, которая одновременно общается со всеми устройствами, входящими в ту же самую область (домен) широковещания или подсеть. В данном случае все устройства в такой области или подсети могут “слушать” сетевую станцию и получать те же самые данные, которые сетевая станция передает всего один раз для всех устройств. Минус такого режима в том, что с помощью видеосервера можно в реальном времени осуществлять трансляцию телепередач, скажем новостей CNN, в сеть масштаба университетского городка, и при этом они будут доступны любому пользователю. В этом случае широковещательный трафик должен пересекать границы подсетей и под него должна быть отведена значительная доля драгоценной пропускной способности. Для поддержки широковещания требуется, чтобы все машины и межсетевое оборудование сети (маршрутизаторы и коммутаторы) обрабатывали пакеты широковещательного трафика, даже если получение широковещательных сообщений требуется только для небольшой части хостов.

Хорошее решение проблемы предлагает групповая адресация. При этом методе связи информация с одной станции может передаваться на несколько станций-приемников одновременно, толдько в отличие от одноадресного и широковещательного режимов, компьютер-передатчик выбирает определенную группу машин для получения его информации. Это становится возможным благодаря групповому адресу, который можно представить себе в виде отдельного телевизионного канала. Машины группы просто "настраиваются" на конкретный групповой адрес для приема через него потока представляющих интерес данных. В данной адресации потоки информации общего доступа передаются по сети лишь один раз и исключительно тем пользователям, которые хотят их получать. Если, например, имеется 40 подсетей, а трансляция CNN принимается только на машины двух из них, то полоса пропускания 38 оставшихся подсетей не используется.

**Классификация и особенности шин в отношении характера передаваемой информации**

**Шина адреса** - компьютерная шина (серия линия, которая соединяет два(более) устройств), используемая для указания физического адреса. В случае, если процессору нужно осуществить чтение(запись) в ячейку памяти, тогда он указывает ее на адресной шине( значение отправляется по шине данных). Система с 32-битной адресной шиной может адресовать ячеек памяти, аналогично для 64-битной системы. Пространство адресуемой памяти составляет 4 ГБ, если каждая ячейка памяти содержит 1 байт.

**Командная шина** предоставляет собой удобный метод инкапсуляции задач, которые приложение должно выполнять в простые и понятные команды. По ней из оперативной памяти поступают различные команды, представление в виде байтов, которые выполняет процессор. Простые команды вкладываются в один байт, но есть и такие команды, для которых требуется 2, 3 и больше байта.

Компьютерная шина, по которой передаются сигналы, определяющие характер обмена информацией по магистрали, наз. **шиной управления.** Сигналы определяют какую операцию требуется производить(считывание или запись), синхронизируют обмен между устройствами и информацией и т.д. В шине управления есть линии, передающие следующие сигналы:

* WR — сигнал записи;
* RD — сигнал чтения;
* MREQ — сигнал инициализации устройств памяти (ПЗУ или ОЗУ);
* IORQ — сигнал инициализации портов ввода-вывода.

**Мультиплексная шина** – это тип структуры шины, в которой количество сигнальных линий, составляющих шину, меньше количества бит данных, адреса или управляющей информации, передаваемых между элементами системы. К примеру, мультиплексированная адресная шина может использовать 8 сигнальных линий для передачи 16 битов адресной информации. Информация передается последовательно(мультиплексируется во временной области, с дополнительными линиями управления, которые используются для последовательной передачи)

2.6. Виртуальные машины. Виртуализация и эмуляция. Гипервизор и супервизор. Контейнеры и докеры.

*Определение 2.6.1*

**Виртуализация** — абстракция вычислительных ресурсов и передача пользователю системы, которая инкапсулирует собственную реализацию. Иначе говоря, пользователь работает с удобным для себя представлением объекта, и для него не играет роли, как объект устроен в действительности.

Одним из видов виртуализации является **виртуализация платформ**

Продуктом этого вида виртуализации являются **виртуальные машины** (virtual machine) - контейнер ПО, который содержит операционную систему и приложения.

Виртуализация платформ - формирование программных систем на основе существующих аппаратно-программных комплексов.

Система, наделяющая аппаратным ресурсом и программным обеспечением, называется **хостовой** (host), а симулируемые ей системы – **гостевыми** (guest). Гостевая система не зависит от архитектуры [хостовой](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82) платформы и реализации платформы виртуализации.

*Определение 2.6.2*

**Гостевая система** - это виртуальная машина, которая использует общие ресурсы, предоставленные хостом. Она содержит приложение и все, что нужно для его запуска: системные исполняемые файлы и библиотеки. Она несет в себе весь аппаратный стек, включая виртуальные сетевые адаптеры, файловое хранилище и центральный процессор, и свою собственную полноценную гостевую ОС.

Процессор с поддержанием виртуализации работает в режимах **root operation** и **non-root operation.**

В режиме root operation работает ПО, являющееся прослойкой между гостевыми операционными системами и оборудованием — монитор виртуальных машин (Virtual Machine Monitor) или гипервизор (hypervisor).

Для перевода процессора в режим виртуализации, платформа виртуализации должна вызвать инструкцию VMXON и отдать управление гипервизору, который запускает виртуальную гостевую систему инструкцией VMLAUNCH и VMRESUME (точки входа в виртуальную машину), см. рис 1.

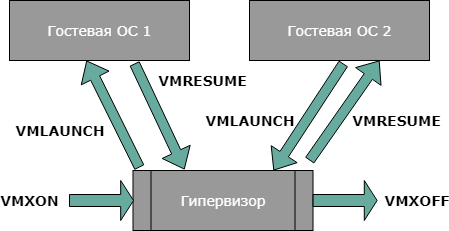


Рис.2.6.1. Перевод в режим виртуализации

Virtual Machine Monitor с помощью вызова инструкции VMXOFF может выйти из режима виртуализации процессора.

Любая из гостевых ОС запускается и работает вне зависимости от других .

#### Аппаратная виртуализация

В аппаратной виртуализации базовым слоем является гипервизор. Этот слой загружается на сервере и обеспечивает взаимодействие между аппаратным обеспечением сервера и виртуальными машинами. Для предоставления ресурсов виртуальным машинам, обеспечивается их виртуализация на сервере. Виртуальные машины запускают свою собственную копию операционной системы и приложений на виртуализированном оборудовании.

##### **Преимущества аппаратной виртуализации**

* Создание множества виртуальных машин с различными операционными системами.
* Виртуальные машины содержат собственное виртуальное оборудование и ПО, запускаемое в виртуальных машинах без модификации.
* Виртуальные машины изолированы друг от друга и от ОС сервера, где происходит запуск виртуальных машин.

**Программная виртуализация на базе ядра ОС**

Программная виртуализация разделяет ресурсы сервера на уровне ядра операционной системы. С помощью средств ОС создаются контейнеры. Виртуальный сервер мало чем отличается от выделенного.

В связи с тем, что все виртуальные машины собраны на одном ядре, пользователь может применять операционную систему, которая совместима с ним. На базе Linux можно устанавливать только дистрибутивы Linux (CentOS, Debian, Ubuntu), что касается Windows, можно использовать только версию, как и у ядра.

**Достоинства программной виртуализации:**

* программная виртуализация не имеет ограничений в отношении архитектуры процессора;
* Возможность увеличить или уменьшить системные ресурсы: оперативную память, дисковое пространство, вычислительные ресурсы процессора;
* Так как на физическом сервере используется только одно ядро, экономятся ресурсы процессора, дисковое пространство.

**Недостатки:**

* Отсутствие разделения ресурсов;
* Использование единой файловой системы, что негативно влияет на общую безопасность;
* Запросы ПО на базе гостевой ОС, обрабатываются сначала общим ядром сервера, затем адресуются к аппаратным ресурсам. В результате процесс исполнения команд замедляется;
* Неосуществимость размещения ресурсоемких проектов.

**Виды виртуализации платформ:**

* **Полная эмуляция (симуляция)**

Проекты, выполняемые по технологии полной эмулиции работают как интерпретаторы: последовательно выбирают код гостевой ОС и эмулируют разные аппаратные архитектуры. Недостаток подхода заключается в высокой потере производительности гостевой ОС.

* **Частичная эмуляция (нативная виртуализация)**

Для изолированного запуска виртуальная машина виртуализирует аппаратное обеспечение, что позволяет запускать гостевые ОС, разработанные только для той же архитектуры, что и у хоста. Таким образом, гостевые системы могут быть запущены параллельно. Этот вид виртуализации увеличивает быстродействие гостевых систем по сравнению с полной эмуляцией.Для быстродействия, в платформах виртуализации, используется «прослойка» между гостевой ОС и оборудованием -гипервизор, который позволяет гостевой системе напрямую обращаться к ресурсам аппаратного обеспечения. Недостатком такого вида виртуализации является зависимость виртуальных машин от архитектуры аппаратной платформы.

**Частичная виртуализация (виртуализация адресного пространства)**

Виртуальная машина симулирует несколько экземпляров аппаратного окружения (но не всего), в частности, пространства адресов. Это позволяет одновременно использовать ресурсы и изолировать процессы, но не разделяет экземпляры гостевых ОС. То есть, пользователем не создаются виртуальные машины, а происходит изоляция каких-либо процессов на уровне ОС.

**Гипервизор и супервизор.**

*Определение 2.6.3*

**Супервизор**–программа, управляющая процессом, памятью и работой оборудования ОС.

*Определение 2.6.4*

**Гипервизор** - программа, позволяющая параллельное выполнение нескольких операционных систем на одном и том же хост компьютере.

Обязанности гипервизора:

* изолирование ОС друг от друга;
* разделение ресурсов между ОС,
* управление ресурcами;
* обеспечение защиты и безопасности ОС.

**Контейнер, докер**

Контейнеры и виртуальные машины имеют схожие цели: изолировать приложение и его зависимости в самостоятельный блок, который можно запускать где угодно.

Основное различие между контейнерами и виртуальными машинами — в их архитектурном подходе.

На рис.2.6.2 представлена архитектура виртуальной машины.



Рис. 2.6.2. Архитектура виртуальной машины

Контейнеры в отличие от виртуальной машины, обеспечивающей аппаратную виртуализацию, контейнер обеспечивает виртуализацию на уровне операционной системы .

Если запуск выполняется в виде контейнера, то приложение попадает в ту же ОС, в которой работают остальные приложения заказчика.

Если запуск осуществляется в виде виртуальной машины на сервере, то любые обращенные в ее сторону запросы со стороны контактирующих приложений проходят длинный путь из одной операционной среды в другую и обратно.

Разница между контейнерами и виртуальными машинами состоит в том, что контейнеры разделяют ядро хоста с другими контейнерами.

На рис.2.6.3 представлена архитектура контейнера

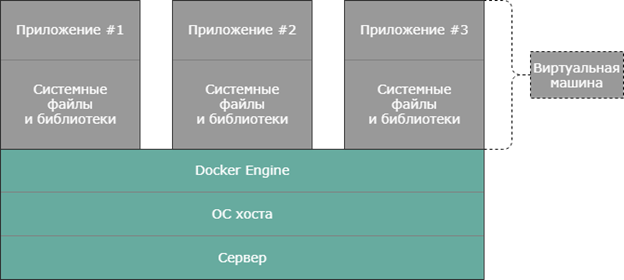


Рис. 2.6.3. Архитектура контейнера

Контейнеры упаковывают только пользовательское пространство, а не ядро или виртуальную аппаратуру, как это делают виртуальные машины. Каждый контейнер получает свое собственное изолированное пользовательское пространство для обеспечения возможности запуска нескольких контейнеров на одном хосте. Архитектура уровня ОС разделяется между контейнерами, именно поэтому контейнеры настолько легковесны.

*Определение 2.6.5*

Docker — ПО для автоматизации развертывания и управления приложениями в среде виртуализации на уровне операционной системы.

**Основные преимущества Docker** :

* Простота использования. Основанная на контейнерах docker платформа позволяет портировать полезную нагрузку. Docker контейнеры могут работать как на реальной локальной машине, так и на виртуальной машине в облаке.
* Скорость работы. Docker-контейнеры легковесны и быстры. Можно создать и запустить Docker-контейнер за секунды, в отличии от виртуальных машин, которые каждый раз запускают полноценную виртуальную ОС.
* Модульность и масштабируемость. Docker позволяет разделять функциональность приложения в отдельные контейнеры.

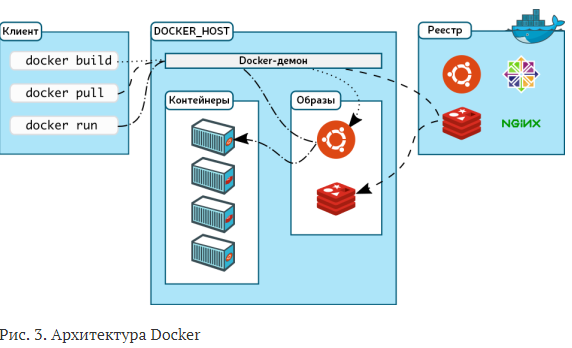


Рис. 2.6.4 Архитектура Docker

2.7. Ядро ОС. Планировщик и механизмы переключения задач

Ядро (англ. Kernel) операционной системы координирует всю ее работу, упорядочивая путь, начиная от системного оборудования до программного и аппаратного обеспечения, которое через графический интерфейс управляется пользователем.

**Функции ядра:**

* обработка прерываний;
* создание и уничтожения процессов;
* переключение процессов из состояния в состояние;
* диспетчирование;
* прерывание и активизация процессов;
* синхронизация процессов;
* организация взаимодействия между процессами;
* манипулирование блоками управления процессами;
* управление устройствами ввода-вывода;
* поддержка выделения и перераспределения памяти;
* поддержка функционирования файловой системы;
* поддержка механизма вызова-возврата при вызове процедур;
* поддержка определенных функций по ведению учета работы машины.

**Варианты реализации**

* **Монолитное ядро.** Широкий набор служебных процедур работает в привелигированном режиме, задействуя общие структуры данных, причем каждая из процедур может вызвать каждую. Так, все компоненты ОС не являются независимыми модулями, а формируют одну большую программу.
* **Модульное ядро.** Предоставляют некоторый механизм подгрузки модулей ядра как динамической, так и статической, поддерживающий определенное аппаратное обеспечение. Таким образом, модульное ядро не требует многократной полной перекомпиляции при каком-либо изменении состава аппаратного обеспечения компьютера.
* **Микроядро**. Обеспечивает работу с оборудованием посредством функций управления процессами и элементарного набора примитивов, реализующих /воплощающих базовые сервисы ос, в которых размещены все или почти все драйверы и модули.
* ОС с **экзоядром** (экзо- — нечто внешнее, находящееся снаружи)  
  Предоставляет функции для межпроцессорного взаимодействия, связанных ограничением — экзоядро должно иметь возможность гарантировать безопасное выделение и освобождение ресурсов оборудования.
* **Наноядро.** Выполняет только задачу обработки аппаратных прерываний, производимых устройствами компьютера и, используя, тот же механизм отправляет отчет о результатах
* **Гибридные ядра**(преобразованные микроядра). Предоставляют возможность ускорить работу запуска второстепенных частей в пространстве ядра.

*Определение 2.7.1*

**Планировщик задач** — часть ОС, отвечающая за построение очереди на выполнение задач, процессов, потоков.

*Определение 2.7.2*

**Невытесняющая планировка**— это способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он по собственной инициативе, не вернет управление планировщику ОС, чтобы тот выбрал из очереди другой, готовый к выполнению процесс.

1. **FIFO** (“первым вошел – первым вышел“). Процесс, ставший готовым к выполнению, присоединяется к очереди готовых процессов. При прекращении выполнения текущего процесса для следующего выполнения отбирается процесс, который находился в очереди дольше других.
2. **EDF** (алгоритм с динамическим планированием задач). Здесь приоритет тем больше, чем меньше срок выполнения. В каждый цикл задачи последовательности выстраиваются заново в зависимости от критического срока выполнения.

*Определение 2.7.3*

**Вытесняющая планировка** –  это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

1. **RM** (алгоритм с фиксированным приоритетом). Приоритет задачи назначается по принципу: чем реже вызывается задача, тем выше ее приоритет.
2. **LSTF** (алгоритм планирования). Приоритет задачи назначается согласно следующему принципу: чем меньше время связывания задачи, тем выше приоритет.

**Механизмы переключения задач**

Нормальное переключение задач обеспечивается специальными дескрипторами  в таблицах дескрипторов LDT, GDT или IDT. Такими дескрипторами являются 16- и 32-разрядные дескрипторы TSS (для запоминания аппаратных контекстов процессов) и дескриптор шлюза задачи.  Дескриптор шлюза задачи предназначен для перенаправления всех указывающих на него вызовов (СALL), переходов (JMP) и векторов прерываний (особых ситуаций) на обработку в отдельную задачу (рис. 2.7.1).

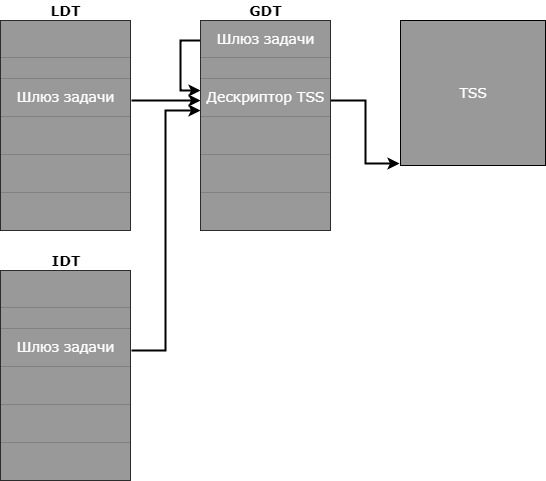


Рис. 2.7.1. Принцип работы механизма переключения задач

 Ниже описаны все выполняемые процессором **действия для переключения задачи**.

1. Получение [селектор](http://www.club155.ru/x86addr-selectors)а [TSS](http://www.club155.ru/x86addr-tss) новой задачи как операнда в командах [CALL](http://www.club155.ru/x86cmd/CALL) и [JMP](http://www.club155.ru/x86cmd/JMP), либо из дескриптора шлюза задачи
2. Проверяется соответствие всех уровней привилегий для определения допустимости переключения на новую задачу.
3. Проверяется [дескриптор](http://www.club155.ru/x86addr-descriptors) [TSS](http://www.club155.ru/x86addr-tss) новой задачи (сегмент присутствует и имеет корректный предел, не менее 67h для 32-битного TSS).
4. Проверяется, что новая задача свободна ([CALL](http://www.club155.ru/x86cmd/CALL), [JMP](http://www.club155.ru/x86cmd/JMP), [особая ситуация или прерывание](http://www.club155.ru/x86exceptions-types)) либо занята ([IRET](http://www.club155.ru/x86cmd/IRET)).
5. Проверка того, что [TSS](http://www.club155.ru/x86addr-tss) текущей (старой) задачи, [TSS](http://www.club155.ru/x86addr-tss) новой задачи и все [дескрипторы](http://www.club155.ru/x86addr-descriptors), используемые при переключении задач присутствуют в памяти.
6. Происходит очистка бита занятости задачи в [TSS](http://www.club155.ru/x86addr-tss) старой задачи, если переключение задач инициировано командами [JMP](http://www.club155.ru/x86cmd/JMP) или [IRET](http://www.club155.ru/x86cmd/IRET) (для [CALL](http://www.club155.ru/x86cmd/CALL), [особой ситуации или прерывания](http://www.club155.ru/x86exceptions-types) бит занятости остается установленным).
7. Сохраняется контекст старой задачи. Процессору требуется знание не только селектора TSS нового процесса, но и селектора TSS текущего процесса (который пока еще находится на ЦП). Для его хранения в ЦП имеется специальный 16-битный регистр – TR (Task Register). Таким образом, процессор определяет базовый адрес [TSS](http://www.club155.ru/x86addr-tss) текущей задачи по значению в [регистре TR](http://www.club155.ru/x86internalreg-system#TR) и затем последовательно сохраняет данные из всех [регистров общего назначения](http://www.club155.ru/x86internalreg-common), [селекторы](http://www.club155.ru/x86addr-selectors) из [сегментных регистров](http://www.club155.ru/x86internalreg-segment), значение [регистра EFLAGS](http://www.club155.ru/x86internalreg-eflags) (причем, если переключение задач инициировано командой [IRET](http://www.club155.ru/x86cmd/IRET), процессор очищает флаг NT в сохраняемой копии [регистра EFLAGS](http://www.club155.ru/x86internalreg-eflags)), значение [указателя команд EIP](http://www.club155.ru/x86internalreg-fpufip) и т.д.   
   На данном шаге процессор проводит переключение задачи. Поэтому, если какая-либо ошибка появляется на шагах с 1 по 7, то переключения задач не происходит и процессор полностью возвращается в прежнее состояние до начала переключения задач. Иначе, процессор заканчивает переключение задач (без некоторых проверок) и генерирует соответствующую ошибку перед началом исполнения новой задачи.
8. Если переключение задач было инициировано командой [CALL](http://www.club155.ru/x86cmd/CALL), [прерыванием или особой ситуацией](http://www.club155.ru/x86exceptions-types), то процессор копирует [селектор](http://www.club155.ru/x86addr-selectors) [TSS](http://www.club155.ru/x86addr-tss) старой задачи в поле LINK [сегмента TSS](http://www.club155.ru/x86addr-tss) новой задачи и фиксирует флаг NT в образе [регистра EFLAGS](http://www.club155.ru/x86internalreg-eflags), хранимом в [TSS](http://www.club155.ru/x86addr-tss) новой задачи.
9. Если переключение задач было инициировано командами [CALL](http://www.club155.ru/x86cmd/CALL), [JMP](http://www.club155.ru/x86cmd/JMP), [прерыванием или особой ситуацией](http://www.club155.ru/x86exceptions-types), процессор определяет флаг занятости B в [дескрипторе](http://www.club155.ru/x86addr-descriptors) [TSS](http://www.club155.ru/x86addr-tss) новой задачи.
10. Фиксируется флаг TS в [регистре CR0](http://www.club155.ru/x86internalreg-control#CR0).
11. Проводится запись в [регистр TR](http://www.club155.ru/x86internalreg-system#TR) значений [селектора](http://www.club155.ru/x86addr-selectors) и [дескриптора](http://www.club155.ru/x86addr-descriptors) (в скрытую часть) [TSS](http://www.club155.ru/x86addr-tss) новой задачи.
12. Проводится чтение контекста новой задачи из [TSS](http://www.club155.ru/x86addr-tss). Считываются значения всех [регистров общего назначения](http://www.club155.ru/x86internalreg-common), [сегментных регистров](http://www.club155.ru/x86internalreg-segment), [регистра флагов EFLAGS](http://www.club155.ru/x86internalreg-eflags), [указателя команд EIP](http://www.club155.ru/x86internalreg-fpufip) и др. (в т.ч. значения для регистров [LDTR](http://www.club155.ru/x86internalreg-system), [CR3](http://www.club155.ru/x86internalreg-control)).
13. Далее выполняются команды новой задачи.

Глава 3. Процессор (программирование)

3.1. Понятие о параллельных и распределенных вычислениях. Программно-аппаратные переходы и аспекты их применения.

*Определение 3.1.1*

**Параллельные вычисления** — способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно.

*Определение 3.1.2*

**Распределенные вычисления** — способ решения больших вычислительных задач с помощью нескольких компьютеров, объединенных в параллельную вычислительную систему (чаще всего).

*Определение 3.1.3*

**Распределенная система** — [система](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), для которой отношения [местоположений](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE) элементов (или групп элементов) играют существенную роль с точки зрения функционирования системы, а так же анализа и синтеза системы.

Для распределенных систем характерно распределение функций, ресурсов между некоторым количеством элементов (узлов) и отсутствие единого управляющего центра, поэтому выход из строя одного из узлов не приводит к полной остановке всей системы. Например, [Интернет](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) это распределенная система.

**Аппаратные и программные средства, реализующие распределенную модель вычислений.**

*Определение 3.1.4*

**Программным средством**, реализующим модель распределенных вычислений, является распределенная операционная система.

Распределенная ОС, динамически и автоматически распределяет работы по разным машинам системы для обработки и заставляет набор сетевых машин работать как виртуальный [процессор](https://en.wikipedia.org/wiki/Uniprocessor_system) (представление ядра физического процессора в ОС логического раздела, использующего общие процессоры). У пользователя распределенной ОС нет информации о том, на какой машине выполняется его работа. Появление сетей, которые предназначены для взаимной связи разных компьютеров, привело к разработке средств и ОС, позволяющих осуществлять управление, мультимашинной архитектурой, то есть совокупности полносоставных компьютеров (процессоры, память, вводы-выводы и т.д.), связанных в сеть.

*Определение 3.1.5*

**Аппаратным средством** модели распределенных вычислений может служить **Грид-вычисления** (англ. grid — решетка, сеть) — это форма распределенных вычислений, в которой «виртуальный суперкомпьютер» представлен в виде кластеров, соединенных с помощью сети слабосвязанных гетерогенных компьютеров, которые работают вместе, чтобы выполнить огромное количество операций. Эта технология применяется для решения научных, математических задач, требующих больших вычислительных ресурсов. Грид-вычисления используются также в коммерческой инфраструктуре для решения таких задач, как экономическое прогнозирование, сейсмоанализ, разработка и изучение свойств новых лекарств.

Грид с точки зрения сетевой организации представляет собой согласованную, открытую и стандартизованную среду, которая обеспечивает гибкое, безопасное, скоординированное разделение вычислительных ресурсов и ресурсов хранения информации, являющихся частью этой среды, в рамках одной виртуальной организации.

**Многопортовая память, зеркалирование памяти, шины с поддержкой режимов групповой адресации (multicasting) и широковещания (broadcasting)**

*Определение 3.1.6*

**Многопортовая память** — это статическое ОЗУ с двумя или более независимыми интерфейсами, которые обеспечивают доступ к пространству памяти через разделенные шины адреса, данных и управления. Структура двухпортового статического ОЗУ (Рис. 3.1.1) содержит один массив памяти (COMMON CENTRAL MEMORY) и два независимых порта (PORT\_L и PORT\_R) для обращения к этому массиву.

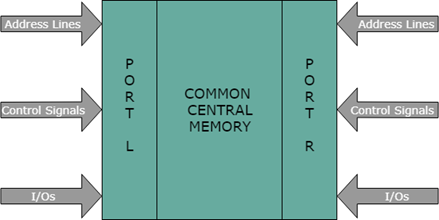


Рис. 3.1.1. Структура двухпортового статического ОЗУ

*Определение 3.1.7*

**Зеркалированние памяти** — это процесс разделения памяти на 2 канала, где первый канал отражается на второй, на втором канале создается избыточная копия памяти. Данный метод позволяет сделать ввод/вывод регистров и памяти в нескольких адресных пространствах, так как один и тот же байт доступен по нескольким адресам.

*Определение 3.1.8*

**Broadcast-шина** — термин, который описывает способ обмена данными, где некоторый объем информации передается из одной точки во все другие точки. Существует лишь один отправитель, информация передается всем соединенным получателям.

*Определение 3.1.9*

**Multicast-шина** — термин, который описывает способ обмена данными, где некоторый объем информации передается из одной или нескольких точек в набор других точек. В данном случае может существовать один или несколько отправителей, а информация распространяется по набору получателей (которых может и не быть).

*Определение 3.1.10*

**Прерывание** - сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении события, требующего немедленного внимания. Прерывание извещает процессор о наступлении события с высоким приоритетом, требующего прерывания текущего кода, выполняемого процессором. Процессор отвечает приостановкой своей текущей деятельности, сохраняя состояние, и выполняет обработчику прерывания, который отвечает на событие и обслуживает его, после чего возвращает управление в прерванный код.

В зависимости от источника возникновения сигнала прерывания делятся на:

* асинхронные (внешние или аппаратные) - события, которые исходят от внешних аппаратных устройств и могут произойти в произвольный момент времени (например, сигнал от таймера, нажатие клавиш клавиатуры, движение мыши). Наличие в системе такого прерывания воспринимается как запрос на прерывание - устройства сообщают, что они требуют внимания со стороны операционной системы;
* синхронные (внутренние) – события, которые происходят при нарушении каких-то условий во время исполнения машинного кода (например, деление на ноль или переполнение стека, обращение к недопустимым адресам памяти или недопустимый код операции);
* программные (частный вид случая внутреннего прерывания) – осуществляются исполнением специальной инструкции в коде программы. Программные прерывания применяются для обращения к функциям встроенного программного обеспечения, драйверов и операционной системы.

Каждому прерыванию присваивается номер, система распознает,какое прерывание с каким номером произошло и запускает соответствующую этому номеру программу обработки прерывания.

Рассмотрим пример прерывания с номером 9 - это прерывание от клавиатуры, которое генерируется при нажатии и при отжатии клавиши. Используется для чтения данных с клавиатуры. Обозначается в ОС как IRQ1, где IRQ – обозначение прерывания, а 1 - приоритет прерывания.

Обработчик прерываний - программа обработки прерывания, которая является частью ОС, предназначенная для выполнения ответных действий на условие, вызвавшее прерывание.

Предположим, что в момент поступления сигнала прерывания от некоторого источника программа А находится в решении. В результате управление автоматически передается обработчику прерываний. После завершения обработки управление может быть снова передано в ту точку программы А, где ее выполнение было прервано:

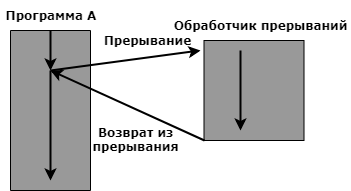


Рис. 3.1.2. Обработчик прерываний

Вектор прерывания – адреса обработчиков прерываний.

Векторы прерываний собираются в таблицу векторов прерываний, включающую в себя адреса обработчиков прерываний. Местоположение таблицы зависит от типа и режима работы процессора.

Таблица векторов прерываний занимает первый килобайт оперативной памяти - адреса от 0000:0000 до 0000:03FF. Таблица состоит из 256 элементов - адресов обработчиков прерываний. В первом слове элемента таблицы записано смещение, а во втором - адрес сегмента обработчика прерывания. Векторами являются просто полные адреса памяти программы ,которая должна быть активизирована в случае возникновения прерывания.

Прерыванию с номером 0 соответствует адрес 0000:0000, прерыванию с номером 1 - 0000:0004 и т.д. Адрес такой состоит из пары 2-байтовых слов, поэтому каждый из векторов занимает четыре байта.

Можно просмотреть таблицу векторов прерываний в компьютере, если воспользоваться программой DЕBUG. Используйте команду D для вывода содержимого начала памяти: D 0:0. Программа DЕBUG покажет вам первые 128 байтов или 32 вектора, которые могут иметь вид наподобие следующего:

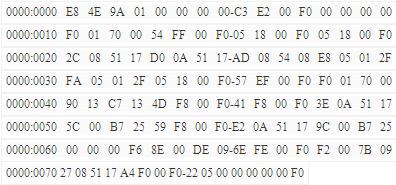


Рис. 3.1.3. Таблица векторов прерываний в компьютере

Обработчик прерываний - процедура, вызываемая по прерыванию для выполнения его обработки.

**Механизм обработки прерываний**

При обработке любого прерывания должны выполняться следующие действия:

* Понимание запроса на прерывание (прием сигнала и распознание прерывания)
* Запоминание состояния прерванного процесса (определяется адресом следующей команды и содержимым регистров процессора)
* Перенос управления прерывающей программе (в счетчик команд заносится начальный адрес подпрограммы обработки прерываний, а в соответствующие регистры - информация из слова состояния процессора).
* Обработка прерывания.
* Восстановление прерванного процесса и возврат в прерванную программу.

**Главные функции механизма прерывания:**

1. Распознавание или классификация прерываний.
2. Передача управления соответственно обработчику прерываний.
3. Правильное возвращение к прерванной программе.

**Типы прерываний**

Прерывания, возникающие при работе вычислительной системы, можно разделить на 4 группы:



Рис. 3.1.4. Типы прерываний

Аппаратные прерывания- прерывания, вызванные процессами в аппаратуре вычислительной системы. Аппаратные прерывания не согласуются с работой программного обеспечения. Когда вызывается прерывание, то процессор оставляет свою работу, выполняет прерывание, а затем возвращается на прежнее место.

Внешние прерывания возникают по сигналу внешнешних устройств.

**Маскируемые и немаскируемые внешние прерывания**

Внешние прерывания подразделяются на два вида: немаскируемые и маскируемые.

Процессор может обладать защитой от прерываний: отключение системы прерываний, маскирование определенных сигналов прерываний.

1. *Маскируемые прерывания* могут быть замаскированы программными средствами компьютера;
2. *Немаскируемые прерывания*, запрос от которых таким образом замаскирован быть не может.

Внутренние прерывания вызываются событиями, которые связаны с работой процессора:

* при нарушении адресации (указан запрещенный или несуществующий адрес);
* существование в поле кода не задействованной двоичной комбинации.
* при делении на ноль.
* при переполнении или исчезновении порядка.
* при обнаружении ошибок четности, ошибок в работе различных устройств аппаратуры средствами контроля.

**Программные прерывания**

Программы могут вызывать прерывания с заданным номером сами. Для этого используется команда INT. Поь команде процессор осуществляет практически те же действия, что и при обычных прерываниях, но только это происходит в определенной точке программы - там, где программист поместил данную команду.

Программные прерывания в прямом смысле прерываниями не являются, поскольку представляют собой лишь специфический способ вызова процедур - не по адресу, а по номеру в таблице.

3.2. Способы передачи параметров в функции и получения данных из функций. Единственность и множественность возвращаемых параметров функции. Описатель restrict.

**Передача параметров**

Обычно, параметры передаются между функциями (написанными на Си или языке ассемблера) через стек. Например, если функция foo1() вызывает функцию foo2() с 2 параметрами (скажем, символами x и y), то перед тем, как control jumps на начало foo2(), два байта (нормальный размер символа в большинстве систем) заполняются значениями, которые должны быть переданы. Когда control jumps на новую функцию foo2(), и вы используете значения (переданные как параметры) в этой функции, они достаются из стека и используются.

Используются две **техники передачи параметров**:

* + - 1. Передача по значению
      2. Передача по адресу

Техники передачи параметров также могут использовать

right-to-left (C-style)

left-to-right (Pascal style)

Можно передавать и слева направо и справа налево.

На процессорах с большим количеством регистров, по стандартным договоренностям вызова помещают все параметры (даже return address) в регистры.

На процессорах с неадекватным количеством регистров, по стандартным договоренностям вызова приходится класть хотя бы некоторые параметры в стек или куда-нибудь еще в оперативную память.

Некоторые договоренности вызова позволяют использовать "reentrant code".

**Передача по значению**

При передаче по значению, передается копия реального значения. Например, если есть функция, принимающая два значения:

void foo (char x, char y) {

x=x+1;  
 y=y+2;  
 putchar(x);  
 putchar(y);  
}

и вы вызываете функцию таким образом:

char a,b;   
 a='A';  
 b='B';  
 foo(a,b);

то программа делает push копий ASCII значений 'A' и 'B' (65 и 66 соответственно) на стек до вызова функции. Можно видеть, что в функции foo() нет упоминания 'a' или 'b'. Таким образом, любые изменения, произошедшие с этими значениями в foo, не будут влиять на значения a и b в вызывающей функции.

**Передача по адресу**

Представьте себе ситуацию, когда вам надо передать в функцию много данных, и применить изменения, произведенные в функции к оригинальным переменным. Примером такой ситуации может служить функция, заменяющая в строке все маленькие буквы на большие (строчные на заглавные). Было бы неразумным передавать всю строку (особенно большую) функции, а когда изменения закончены, возвращать такой же большой результат. Здесь мы передаем адрес переменной в функцию. У этого метода есть два преимущества: нет необходимости передавать много данных, что сокращает время исполнения, и появляется возможность работать с данными напрямую, таким образом, к концу функции все данные из вызывающей функции уже модифицированы.

Но надо помнить, что любые изменения переменной, переданной по адресу происходят сразу с оригинальной переменной. Если этого не хочется, то надо вручную скопировать переменную перед ее изменением.

В языке программирования C, начиная со стандарта C99, restrict - это ключевое слово, которое можно использовать в объявлениях указателей. Оно дает возможность сообщить компилятору, что объявляемый указатель указывает на блок памяти, на который не указывает никакой другой указатель.

Если компилятор знает, что только один указатель указывает на блок памяти, то он может создать более оптимизированный код. Например,

void updatePtrs ( size\_t \* ptrA, size\_t \* ptrB, size\_t \* val )

{ \*ptrA += \*val; \*ptrB += \*val; }

Указатели ptrA, ptrB и val могут указывать на один и тот же блок памяти.

Для этой функции компилятор будет генерировать примерно следующий код:

*; прочитать значение из памяти по указателю val*

load R1 ← \*val

*; прочитать значение из памяти по указателю ptrA*

load R2 ← \*ptrA

*; выполнить сложение*

add R2 += R1

*; записать результат в память по указателю ptrA*

set R2 → \*ptrA

*; аналогично для ptrB*

load R1 ← \*val *; чтение по val второй раз*

load R2 ← \*ptrB

add R2 += R1

set R2 → \*ptrB

Значение по указателю val считывается из памяти дважды, так как указатель ptrA может указывать на тот же блок памяти, что и val.

При использовании ключевого слова restrict определение функции будет следующим:

void updatePtrs (

size\_t \* **restrict** ptrA,

size\_t \* **restrict** ptrB,

size\_t \* **restrict** val

) {

\*ptrA += \*val;

\*ptrB += \*val;

}

Здесь restrict сообщает компилятору, что указатели ptrA, ptrB и val никогда не указывают на один и тот же блок памяти. Это гарантируется программистом.

Тогда компилятор будет генерировать примерно следующий код:

load R1 ← \*val

load R2 ← \*ptrA

add R2 += R1

set R2 → \*ptrA

*; load R1 ← \*val ; отсутствует*

load R2 ← \*ptrB

add R2 += R1

set R2 → \*ptrB

3.3.Компиляция и интерпретация исходных текстов программ. Объектный код, исполняемый код, динамическое и статическое связывание модулей программы. Байт-код.

**Компиляция и интерпретация исходных текстов программ: сравнительный анализ механизмов и возможностей.**

Для того, чтобы программа исполнялась компьютером, ее необходимо привести к понятному для «машины» виду.

Имеется два способа получения программы в пригодном для исполнения виде:

* компиляция;
* интерпретация.

*Определение 3.3.1*

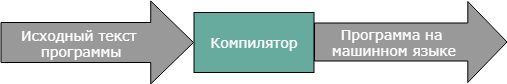
**Компилятор** (англ. compiler - составитель, собиратель) читает программу целиком, осуществляет ее перевод и создает законченный вариант программы на машинном языке, который вслед за этим выполняется. Результатом работы компилятора является бинарный исполняемый файл.

*Определение 3.3.2*

**Интерпретатор** (англ. interpreter - истолкователь, устный переводчик) переводит и выполняет программу строка за строкой.

*Определение 3.3.3*

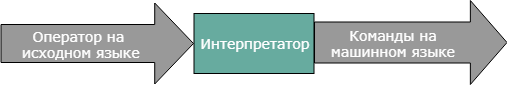
**Компиляцией** **исходных текстов** называется процесс преобразования программы, написанной на одной языке, в полностью эквивалентную программу на другом языке.

**

*Рис.3.3.1. Компиляция исходных текстов*

*Определение 3.3.4*

**Интерпретация исходного текста программы** – это процесс ее выполнения программой-интерпретатором последовательно, оператор за оператором.

**

*Рис.3.3.2. Интерпретация исходного текста программы*

Алгоритм работы простого интерпретатора:

1. Прочитать инструкцию;
2. Проанализировать инструкцию и определить соответствующие действия;
3. Выполнить соответствующие действия;
4. В том случае, если не достигнуто условие завершения программы, то прочитать следующую инструкцию и перейти к пункту b.

**Основные различия компиляции и интерпретации:**

* Интерпретатор берет одну инструкцию, транслирует и выполняет ее. Потом берет следующую инструкцию, а компилятор транслирует всю программу сразу, после чего выполняет ее.
* Компилятор генерирует отчет об ошибках после трансляции всего, в то время как интерпретатор прекратит трансляцию после первой найденной ошибки.
* Компилятор требует больше времени для анализа и обработки языка высокого уровня, чем интерпретатор.
* Помимо времени на обработку и анализ, общее время выполнения кода компилятора в сравнении с интерпретатором будет быстрее.

**Понятие об объектном коде, исполнимом коде, динамическом и статическом связывании модулей программы.**

*Определение 3.3.5*

Низкоуровневое представление компьютерной программы, чаще всего не в форме, непосредственно пригодной для объектного компьютера, а в форме, включающей в себя информации о процессорных командах, информацию о размещении программы, называется **объектным кодом**.

Исходный код транслируется в **исполнимый код** с помощью компилятора до начала запуска программы, но также он может исполняться сразу с помощью интерпретатора.

Динамическое и статическое связывание - это методы обмена библиотечным кодом в современных компьютерных операционных системах.

*Определение 3.3.6*

**Динамическое связывание** происходит во время выполнения. Он запускает небольшую дополнительную работу во время запуска для некоторых хороших функций, таких как возможность исправления/замены библиотеки, которая, в свою очередь, может исправлять или добавлять функции к приложениям, которые связаны (привязаны) к ней.

*Определение 3.3.7*

**Статическое связывание** - метод, где части библиотеки копируются в исполняемый файл. Этот метод в сравнении с динамическим связыванием быстрее для запуска, а в некоторых случаях может быть более удобным для распространения и может сэкономить место. Оно обеспечивается на стадии формирования исполнительного модуля, еще до этапа выполнения программы.

**Байт-код: его назначение, получение, оптимизация и выполнение.**

Байт-код , также называемый переносимым кодом или p-кодом , является формой набора команд, разработанного для эффективного выполнения программным интерпретатором. В отличие от читаемого человеком исходного кода , байт-коды представляют собой компактные числовые коды, константы и ссылки (обычно числовые адреса), которые кодируют результат анализа компилятора и выполнения семантического анализа таких вещей, как тип, область действия и глубина вложенности программных объектов.

Сначала исходный текст программы компилируется. Результат этой компиляции - это некий условный машинный код, который не зависит от вычислительной архитектуры и может быть выполнен на разных операционных системах, разными процессорами и его результат все равно будет корректным. Далее этот байт-код запускается на некотором подобии виртуальной машины, то есть эта программа, которая принимает в себя операции на байт-коде и переводит в последовательность команд данной архитектуры процессора и вызовов к данной операционной системе. Таким образом происходит интерпретация байт-кода на конкретной платформе, превращая этот код в исполнимые на данном процессоре действия.

Разные современные языки программирования(особенно интерпретируемые) используют байт-код для ускорения и облегчения работы интерпретатора. Трансляция в байт-код является методом, промежуточным по эффективности между прямой интерпретацией и компиляцией в машинный код.

Свое название байт-код получил в связи с тем, что длина каждого кода операции составляет один байт и каждая инструкция обычно представляет собой однобайтовый код операции, за которым могут следовать различные параметры (номер регистра / адреса в памяти).

Программа байт-кода может быть выполнена путем анализа и непосредственного выполнения инструкций, по одной за раз. Преимущество байт-кода в большей эффективности и портируемости, то есть один и тот же байт-код может исполняться на разных платформах и архитектурах, для которых реализован интерпретатор. Эффективность интерпретации байт-кода обычно выше, чем чистая интерпретация исходного кода или интерпретация АСД. Кроме того, интерпретатор байт-кода чаще всего проще интерпретатора исходного кода и его легче перенести на другую аппаратную платформу.

Прямая оптимизация байт-кода: некоторые инструкции байт-кода намного трудозатраные, чем другие. Например, загрузка локальной переменной в стек недолгая, но вызовы виртуальных методов, вызовы интерфейса, распределение объектов и перехват исключений идет затратнее. Традиционные C-подобные оптимизации, такие как копирование, имеют незначительный эффект, потому что они не нацелены на дорогие байт-коды. Выполнять эффективно оптимизации на этом уровне, необходимо рассмотреть более продвинутые оптимизации, такие как встраивание методов и статическое разрешение вызовов виртуальных методов, которые напрямую уменьшают использование этих дорогих байт-кодов.



Рис.3.3.3. Принцип работы байт-кода

3.4 Специализированные средства разработки программного обеспечения

*Определение 3.4.1*

**Ассемблеры** — компьютерные программы, реализующие преобразование программы в форме исходного текста на языке ассемблера в машинные команды в виде объектного кода.

*Определение 3.4.2*

**Трансляторы** — технические средства, выполняющие преобразование программы, написанной на одном из языков программирования, в бинарный файл программы, который состоит из машинных команд, либо выполняющие непосредственно действия программы.

*Определение 3.4.3*

**Препроцессоры исходных текстов** — это компьютерные программы, обрабатывающие текст исходной программы до компилятора. С помощью препроцессора возможно выполнение следующих операций:

* включение в программу текстов из указанных файлов;
* замена идентификаторов последовательностями символов;
* макроподстановка (замена обозначения параметризованным текстом, формируемым препроцессором с учетом конкретных аргументов);
* исключение из программы отдельных частей текста (условная компиляция)

*Определение 3.4.4*

**Отладчик** является модулем среды разработки, предназначенным для поиска ошибок в программе.

*Определение 3.4.5*

**Текстовые редакторы** — компьютерные программы для создания и изменения текстовых файлов, а также их просмотра на экране, вывода на печать, поиска фрагментов текста.

*Определение 3.4.6*

**Специализированные редакторы исходных текстов** — текстовые редакторы для создания и редактирования исходного кода программ.

*Определение 3.4.7*

**Библиотеки подпрограмм** — сборники подпрограмм или объектов для использования в качестве целых частей при составлении новых программ. Библиотека подпрограмм обеспечивает ускорение и облегчает программирование задач.

*Определение 3.4.8*

**Профилирование** — сбор характеристик работы программы:

* выполнение подпрограмм;
* число условных переходов;
* число кэш-промахов.

Инструмент, анализирующий работу, называют профилировщиком или профайлером (profiler).

Благодаря отладке и профилированию можно обнаружить тормозящую функцию, переписать ее на C, вместо C++, а затем тем же способом найти ее самую тормозящую часть и переписать ее на языке ассемблера.

*Определение 3.4.9*

**Точка останова** (англ. breakpoint) — это преднамеренное прерывание выполнения программы, осуществляющее вызов отладчика (параллельно программа может использовать точки останова для собственного применения). После остановки в отладчике программа может быть завершена либо продолжена с места остановки.

Точка останова определяется как условие, при котором прерывается программа. Часто используется условие останова при переходе управления к указанной инструкции программы (instruction breakpoint). Также условием останова может быть операция чтения, записи или изменения указанной ячейки, или диапазона ячеек памяти (data breakpoint или watchpoint*)*.

Немало процессоров включают аппаратную поддержку точек останова. В случае отсутствия отладчики используют программные точки останова.

Для выполнения точек останова используется специальная инструкция процессора. Выполнение этой инструкции приводит к исключению *EXCEPTION\_BREAKPOINT*. Установить точку останова можно в любом месте программы, записывая по соответствующему адресу инструкцию int 3 (1 байт с кодом 0xCC). Разумеется, что это можно сделать и вручную, вставив в программу инструкцию asm int 3.

Метка указывает ассемблеру, что необходимо создать переменную с этим именем, содержащую адрес команды, идущей после метки. Метка в современном ассемблере мало отличается от метки в С. Название метки начинается с буквы (ни в коем случае не с цифры!). Метки используются при прыжках (jmp) или с командой call.

Пример использования метки для того, чтобы перепрыгнуть процедуру, которую мы хотим реализовать в теле другой процедуры:

|  |  |
| --- | --- |
| .code  start  proc  jmp ml  myproc  proc near  ret  myproc  endp  ml:  ...  start  endp  end start | Начало кода программы  В этой (главной) процедуре мы хотим сделать еще одну процедуру  Перепрыгиваем процедуру, которую мы делаем (безусловный переход)  Наша процедура  Она выполняет всего одну команду - ret      Конец ее описания  Это метка, сюда прыгаем  ... Теперь мы можем вызывать нашу процедуру  Конец главного процесса  Указание на то, что программа начинается с метки start |

Первым символом в метке должна быть буква или спецсимвол. Цифра не может быть первым символом метки, а символы “*$”* и “*?”* не рекомендуются к использованию. К регистру не чувствителен, но различие можно включить, путем задания той или иной опции в командной строке ассемблера. Максимальная длина метки - 31 символ. Примеры меток:COUNT, PAGE25, $E10. Рекомендуется использовать описательные и смысловые метки. Имена регистров, например, AX, DI или AL являются забронированными и применяются только для указания соответствующих регистров.

Если метка располагается перед командой процессора, сразу после нее всегда ставится символ *«:»* (двоеточие), указывая ассемблеру, что необходимо создать переменную с этим именем, содержащую адрес текущей команды:

some\_loop:

lodsw; cчитать слово из строки,

cmp ax,7; если это 7 - выйти из цикла

loopne some\_loop

Если метка стоит перед директивой ассемблера, то она оказывается одним из операндов этой директивы и двоеточие не ставится:

codesg segment

lodsw; cчитать слово из строки,

cmp ax,7; если это 7 - выйти из цикла

codesg ends

Рассмотрим директивы, которые работают напрямую с метками и их значениями:LABEL*,* EQUи =.

**1) Директива LABEL**

метка label тип

ДирективаLABEL определяет метку и задает ее тип. Тип может быть одним из:

* BYTE (байт),
* WORD (слово),
* DWORD (двойное слово),
* FWORD (6 байт),
* QWORD (учетверенное слово),
* TBYTE (10 байт),
* NEAR (ближняя метка),
* FAR (дальняя метка).

Метка получает значение, равное адресу последующей команды или следующих данных, и тип, указанный явно. В зависимости от типа команда

mov метка,0

запишет в память байт (слово, двойное слово и т.д.), заполненный нулями, а команда

call метка

выполнит ближний или дальний вызов подпрограммы.

С помощью директивы LABELрационально организовывать доступ к одним и тем же данным, как к байтам, так и к словам, определив перед данными две метки с разными типами.

**2) Директива EQU**

Директива EQU присваивает метке значение, которое определяется как результат целочисленного выражения в правой части. Результатом этого выражения есть целое число, адрес или строка символов:

метка equ выражение

truth equ 1

message1 equ 'Try again$'

var2 equ 4[si]

cmp ax,truth ; cmp ax,1

db message1 ; db 'Try again$'

mov ax,var2 ; mov ax, 4[si]

Директива *EQU* используется с целью введения параметров, общих для всей программы, аналогично команде #define препроцессора языка С.

**3) Директива =**

Директива = равносильна EQU, но определяемая ею меткапринимает только целочисленные значения, которая может быть переопределена.

Каждый ассемблер предлагает набор специальных предопределенных меток — это может быть текущая дата (@date или ??date), тип процессора (@cpu) или имя сегмента программы, но единственная предопределенная метка, поддерживаемая всеми рассматриваемыми нами ассемблерами, — $. Она всегда соответствует текущему адресу.

Например, команда

jmp $

выполняет переход на саму себя, так что создается вечный цикл из одной команды.

3.9. Загрузчики и компоновщики. Форматы объектных модулей. Оверлейные загрузчики и оверлеи.

*Определение 3.9.1*

**Загрузчик** — программа, которая отвечает за загрузку исполнимых файлов и запуск надлежащих последних процессов. Как правило считается составляющей операционной системы, однако способен быть и независимой программой, например, позволяющей ОС запускать программы, скомпилированные для иных ОС.

При запуске новой программы загрузчику нужно:

* Считать данные из запускаемого файла.
* Если нужно — загрузить в память отсутствующие динамические библиотеки.
* Заменить в коде новой программы относительные адреса и символические ссылки на точные, с учетом текущего размещения в памяти, то есть осуществить соединение адресов.
* Создать в памяти образ нового процесса и запланировать его к выполнению.

Загрузчик ОС функционирует согласно аналогичному принципу, однако как правило считается отдельной программой, так как решает особую задачу — запуск самой ОС.

*Определение 3.9.2*

**Компоновщик** (также **редактор связей** или **линкер**) — это инструментальная программа, которая совершает **компоновку** (*«***линковку***»*): принимает на вход один или несколько объектных модулей и составляет по ним исполнимый модуль.

Для связывания модулей компоновщик пользуется таблицами символов, которые формируются компилятором в каждом из объектных модулей.

*Определение 3.9.3*

**Таблица символов** — это структура данных, применяемая транслятором (компилятором или интерпретатором), в которой каждый идентификатор переменной или функции из исходного кода ассоциируется с информацией, связанной с его объявлением либо появлением в коде: типом данных, областью видимости и в отдельных случаях местом в памяти.

Данные таблицы могут содержать символы следующих типов:

* Определенные либо экспортируемые имена — функции и переменные, определенные в данном модуле и предоставляемые для использования иным модулям;
* Неопределенные или импортируемые имена — функции и переменные, на какие ссылается модуль, однако никак не определяет их внутри себя;
* Локальные — могут применяться внутри объектного файла в целях упрощения процесса настройки адресов.

Для большинства компиляторов, один объектный файл является итогом компиляции одного файла с исходным кодом. В случае если программа составляется из нескольких объектных файлов, компоновщик собирает эти файлы в общий исполнимый модуль, вычисляя и подставляя адреса вместо символов, в течение времени компоновки(статическая компоновка) или во время исполнения (динамическая компоновка).

Компоновщик способен извлекать объектные файлы из специальных коллекций, именуемых библиотеками. Если не все символы, на которые ссылаются пользовательские объектные файлы, определены, то компоновщик ищет их определения в библиотеках, которые пользователь передал ему на вход. В большинстве случаев, одна либо некоторое количество системных библиотек используются компоновщиком по умолчанию. Когда объектный файл, в котором находится определение какого-то искомого символа, обнаружен, компоновщик может включить его в исполнимый модуль (в случае статической компоновки) или отложить это до момента запуска программы (в случае динамической компоновки).

Функция компоновщика состоит в том, чтобы во всяком модуле определить и соединить ссылки на неопределенные имена. Для каждого импортируемого имени есть его определение в других модулях, упоминание имени заменяется на его адрес.

Компоновщик как правило не осуществляет проверку типов и количества параметров процедур и функций. Если необходимо объединить объектные модули программ, написанные на языках со строгой типизацией, то требуемые проверки обязаны быть выполнены дополнительной утилитой перед запуском редактора связей.

*Определение 3.9.4*

**Объектный модуль** — файл с промежуточным представлением отдельного модуля программы, полученный вследствие обработки исходного кода компилятором. Объектный файл включает в себя специальным образом подготовленный код (зачастую называемый двоичным или бинарным), который может быть объединен с прочими объектными файлами с помощью компоновщика для получения готового исполнимого модуля или библиотеки.

*Определение 3.9.5*

**Исполнимый (исполняемый) модуль, исполняемый файл** — файл, который содержит программу в том виде, в котором она может быть выполнена компьютером. Перед исполнением программа загружается в память, и производятся определенные подготовительные операции.

Объектные файлы представляют собой блоки машинного кода и данных с неопределенными адресами ссылок на данные и процедуры в других объектных модулях, и кроме того перечень собственных процедур и данных. Компоновщик собирает код и данные каждого объектного модуля в итоговую программу, вычисляет и заполняет адреса перекрестных ссылок между модулями. Соединение со статическими библиотеками производится редактором связей либо компоновщиком (может быть отдельной программой или частью компилятора), а с ОС и динамическими библиотеками связывание производится при выполнении программы после ее загрузки в память.

**Форматы объектных модулей.** Файл объектного модуля содержит отдельные секции данных, каждая из которых включает в себя конкретный тип данных:

* Заголовок (описание файла и нужные компоновщику сведения)
* Сегмент кода (исполнимый код)
* Сегмент данных (инициализация переменных)
* Сегмент констант (инициализация констант, описание глобальных переменных)
* Внешние определения и ссылки на связи
* Информация о динамических связях
* Отладочная информация

*Определение 3.9.6*

**Overlay** — метод программирования, который позволяет создавать программы, занимающие больше памяти, чем установлено в системе.

Этот способ подразумевает деление программы на части, которые называются оверлеями (overlays). Объем каждого оверлея ограничен, в соответствии с размером доступной памяти. Место в памяти, куда будет загружен оверлей называется регионом (region, destination region). Несмотря на то, что зачастую программы используют лишь один блок памяти для загрузки разных оверлеев, допустимо определение нескольких регионов разного размера. Загрузчик оверлеев, в некоторых случаях являющийся частью операционной системы, подгружает запрашиваемый оверлей из внешней памяти в регион, обеспечивает все нужные связи, к примеру, с библиотеками.

**Приложение**

**Специализированные алгоритмы и вычисления с повышенной точностью.**

Существует задачи: вычисления криптографических ключей, задачи астрономии и др., которые требуют большой точности. Точность целых чисел измеряется до . Для чисел с плавающей точкой необходимо изобразить много разрядов целых чисел и большую мантиссу. Существует большое количество библиотек, таких как Bignum, которые выстраивают эти данные друг за другом. И есть специальные разработанные алгоритмы, особенность которых – использование нетривиальных констант, заданных с фиксированной точностью, позволяющие работать с такими кортежами, интерпретируя их как число либо целое, либо с плавающей точкой. Процесс занимает длительное время, но повышает точность операции. Рунге-Ромберг, Ньютон, Лакс-Вендорф, дихотомия – частные случаи, но все же реализующие невысокую точность по сравнению с необходимой.

**Представление данных с фиксированной точкой и правила вычислений и преобразования.**

q - основание системы счисления

r - количество разрядов под дробную часть

n - количество разрядов под целую часть

**,** где - коэффициенты

**,**

При представлении числа в форме с фиксированной точкой указываются знак числа (*sign a*) и модуль числа (*mod a*) в q-ичном коде. Место точки (запятой) постоянно для всех чисел и в процессе решения задач не меняется. Знак положительного числа кодируется цифрой «0», а знак отрицательного числа — цифрой «1».

*Определение 1.5.4*

Код числа в форме с фиксированной точкой, состоящий из кода знака и q-ичного кода его модуля, называется **прямым кодом**.

*Определение 1.5.5*

Разряд прямого кода числа, в котором располагается код знака, называется знаковым **разрядом кода**.

Разряды прямого кода числа, в которых располагается q-ичный код модуля числа, называются цифровыми разрядами кода. При записи прямого кода знаковый разряд располагается левее старшего цифрового разряда и обычно отделяется от цифровых разрядов точкой.

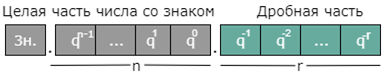


Рис. 1.5(6).1. Разрядная сетка компьютера для размещения чисел в форме с фиксированной точкой

На рисунке показано n разрядов целой части числа и r разрядов — для дробной части числа. Использование формы с фиксированной точкой для представления чисел с целой и дробной частью в компьютерах не используется. В основном, используются компьютеры либо с дробной арифметикой (n=0), либо с целочисленной арифметикой (r=0).

Форма представления чисел с фиксированной точкой упрощает аппаратную реализацию компьютера, уменьшает время выполнения машинных операций, однако при решении задач необходимо постоянно следить за тем, чтобы все исходные данные, промежуточные и окончательные результаты находились в допустимом диапазоне представления. Если этого не соблюдать, то возможно переполнение разрядной сетки, и результат вычислений будет неверным. От этих недостатков в значительной степени свободны компьютеры, использующие форму представления чисел с плавающей точкой.

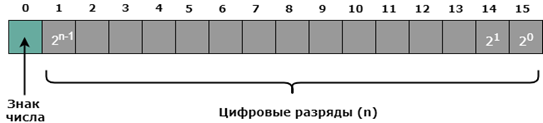


Рис. 1.5(6).2. Пример числа с фиксированной точкой. Ячейка с записью целого числа

К достоинствам использования чисел с фиксированной точкой относятся простота выполнения алгоритмов арифметических операций и высокая точность представления чисел. К недостаткам – небольшой диапазон представления чисел.

Пример числа с плавающей точкой:

2,62510 = 10,1012 = 0,101012\*2210 (нормальная форма)

Если число х положительное, то его прямой код представлен как х2.

Если число х отрицательное, то его прямой код представлен как 1 - х2.

Правила вычислений:

Сложение

Операции сложения и умножения обычно выполняются над двоичными, числами, являющимися правильными дробями. Последовательность выполнения операции сложения, следующая:

* исходные числа записываются в принятом для данной машины коде;
* производится поразрядное сложение кодов чисел, включая и знаковые разряды;
* производится анализ на переполнение разрядной сетки. В случае переполнения поступает сигнал на прерывание программы.

Сложим числа: x=0,101001, y=0,011011.

Сочетание 01 в знаковых разрядах свидетельствует о переполнении.

Умножение

Умножение производится в столбик, как обычное умножение: путем сложения со сдвигами и последующим сложением этих чисел. Знак результирующего числа определяется путем сложения знаков изначальных чисел.

Пример: x=0,1101, y=1,1011. (Т.е. x - положительное число, у - avxотрицательное). Знак их произведения: 0 + 1 = 1 ⇒ число отрицательное.

Ответ: ху = 1,1000111. до этого момента в оптимизацию или в приложение