Оглавление

[Глава 1. Вычислители 2](#_Toc2693586)

[1.1. Электронная вычислительная машина. Классификация ЭВМ 2](#_Toc2693587)

[1.2. Архитектура вычислителя и его расширяющие компоненты 9](#_Toc2693588)

[1.3. Вычислители с множественными потоками команд и одиночным потоком данных (MISD), истинно параллельные процессоры (MIMD), систолические вычислители. 32](#_Toc2693589)

[1.4. Специализированные вычислители. 38](#_Toc2693590)

[1.5. Графический процессор 42](#_Toc2693591)

[1.6. Программная среда специализированного вычислителя 44](#_Toc2693592)

[Глава 2. Процессор (компьютер) 47](#_Toc2693593)

[2.1. Узлы процессора 47](#_Toc2693594)

[2.2. Ядро процессора 57](#_Toc2693595)

[2.3 Прерывания 66](#_Toc2693596)

[2.4. Представление о микросхемах обвязки процессоров 73](#_Toc2693597)

[2.5. Представление о конвейерном выполнении команд в процессорах 76](#_Toc2693598)

[2.6. Классификация и особенности шин в отношении способа передачи, метода синхронизации, топологии, способа управления, адресации устройств 84](#_Toc2693599)

[2.7. Виртуальные машины. Виртуализация и эмуляция. Гипервизор и супервизор. Контейнеры и докеры. 93](#_Toc2693600)

[2.8. Ядро ОС. Планировщик и механизмы переключения задач 102](#_Toc2693601)

[Глава 3. Процессор (программирование) 106](#_Toc2693602)

[3.1. Понятие о параллельных и распределенных вычислениях. Программно-аппаратные переходы и аспекты их применения. 106](#_Toc2693603)

[3.2.Компиляция и интерпретация исходных текстов программ. Объектный код, исполняемый код, динамическое и статическое связывание модулей программы. Байт-код. 111](#_Toc2693604)

[3.3. Специализированные средства разработки программного обеспечения 115](#_Toc2693605)

[3.4. Загрузчики и компоновщики. Форматы объектных модулей. Оверлейные загрузчики и оверлеи. 122](#_Toc2693606)

[Глава 4. Параллельные вычисления 126](#_Toc2693607)

[Приложение 131](#_Toc2693608)

[Список сокращений 135](#_Toc2693609)

# 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 классифицируются 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, то есть 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 по условиям эксплуатации:

* офисные – рассчитаны для решения обширного спектра задач в нормальных условиях эксплуатации.
* специальные – предназначены для решения узкого круга задач либо одной задачи многократного решения. Применяются в автомобилях скорой помощи, на ракетах, самолетах и вертолетах, 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 определяемый круг задач осуществляется наиболее результативно.

Классификация компьютеров в отношении 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.

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 1.1.3.*

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 1.1.4.*

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

На рисунках ниже присутствуют три типа шин. При помощи адресной шины (см. *Определение 2.6.10*) формируется указатель на требуемые данные, а шина данных передает их компонентам компьютера. По шине управления (см. *Определение 2.6.12*) передаются сиг­налы, которые определяют характер обмена информацией, например, сиг­налы чтения, записи, готовности.

*Определение 1.1.5.*

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), определения которых представлены ниже.

*Определение 1.1.6.*

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

*Определение 1.1.7.*

Арифметико-логическое устройство (АЛУ) — блок процессора, который под управлением устройства управления 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).

*Определение 1.1.8.*

Регистры *—* это специальные ячейки памяти, расположенные непосредственно в процессоре. Работа с регистрами выполняется намного быстрее, чем с ячейками оперативной памяти (см. *Определение 1.2.9*), 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.

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

* извлечение очередной команды из памяти;
* 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 содержит важные регистры для хранения информации, необходимой в ходе выполнения текущей команды.

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

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

Выделяют два основных узла компьютера: центральный процессор и память компьютера.

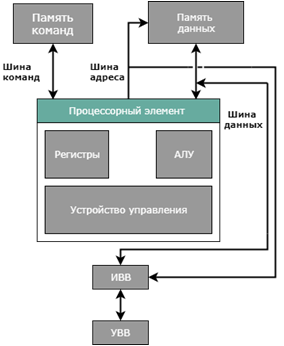
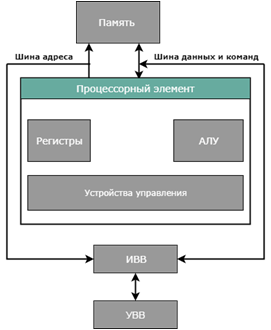
****

Рис.1.1.1. Принстонская архитектура Рис.1.1.2. Гарвардская архитектура

Принстонская архитектура (фон Неймана) характеризуется использованием общей оперативной памяти для хранения программ, данных, а также для организации стека. Для обращения к этой памяти используется общая системная шина (см. *Определение 2.2.8.*), по которой в процессор поступают и команды, и данные.

Классическая гарвардская архитектура 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.

По производительности и характеру использования компьютеры можно условно подразделить на:

*Определение 1.1.9.*

Микрокомпьютер — компьютер, полностью расположенный на одной микросхеме.

Продвинутые модели микрокомпьютеров имеют несколько процессоров. Производительность компьютера определяется еще и емкостью оперативной памяти, типами периферийных устройств (см. *Определение 1.2.13*), качеством конструктивных решений и др.

*Определение 1.1.10.*

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

*Определение 1.1.11.*

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

*Определение 1.1.12.*

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

*Определение 1.1.13.*

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

Можно выделить следующие группы в классе компьютеры:

* мэйнфреймы — распределенные компьютеры, в том смысле, что разные узлы одного компьютера, в силу своих размеров, оформлены в виде отдельных блоков. Применяются в работе с активными вычислениями и данными массивных размеров.
* миникомпьютеры — компьютеры небольших размеров, обладающие более низкими параметрами, чем мэйнфреймы, к которым 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:
* 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 (см. *Определение 1.2.5*);
* системы управления и ввода/вывода информации.

1.2.1. Кэш

Понятие о «горячих» и «холодных» данных.

Разные данные используются пользователями с разной частотой. Например, для ежедневной отчетности и оперативного анализа нужны данные за последние несколько дней, в то время как обращение к данным, которые старше года, требуется не каждый день, и по малому количеству показателям. Характеристику данных через частоту обращения к ним легко представляется в виде температуры, как в школьной физике, только вместо молекул здесь «движутся» блоки данных. Таким образом, данные, лежащие без малейшего движения на текущем временном интервале, называют холодными, а наиболее востребованные – горячими.

*Определение 1.2.2.*

Кэш-память — высокоскоростная память произвольного доступа, не имеющая адресов и используемая процессором компьютера для временного хранения данных.

Кэш-память служит своеобразным буфером между различными устройствами для временного хранения и обработки данных. 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:

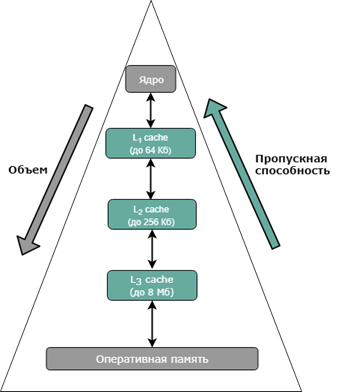
****

Рис. 1.2.1. Уровни кэш-памяти

* Кэш первого уровня (L1) — наиболее быстрый и наименьший по объему, имеет 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.
* 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.
* Кэш третьего уровня (L3)  – в массовых процессорах используется недавно. Он медленнее (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 компонент кэш-памяти кроме байт данных оперативной памяти должен сохранять и его четырехбайтовый адрес в оперативной памяти. Такая своего рода строка называется кэш-строкой(*cache-line*).

В кэше всякой из кэш-строк дополнительно назначается в соответствие тег (*tag*), являющийся адресом повторяющихся в этой кэш-строке данных в основной памяти. Помимо 24 бит тег имеет бит значимости, проверяющий действительность строки (когда бит значимости 0, данная строка принимается недействительной и обращения к данным не кэшируются).

При доступе процессора в память прежде всего проверяется, содержит ли кэш требуемые из памяти данные. Для этого совершается сопоставление адреса запроса значениям всех тегов кэша, в которых запрашиваемые данные могут храниться. Если есть совпадение с тегом некоторой кэш-строки — это называется попаданием в кэш (*cache hit*), тогда осуществляется чтение или запись данных в этой кэш-строке, иная ситуация именуется кэш-промахом (*cache miss*).

Отношение числа попаданий в кэш к общему числу запросов к памяти называют рейтингом попаданий (*hit rate*), определяет меру производительности кэша для выбранного алгоритма или программы.

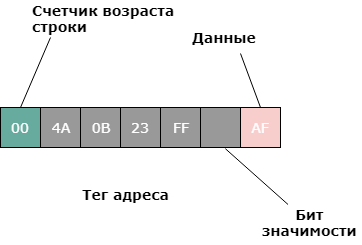
****

Рис. 1.2.2. Структура гипотетического кэша

Загрузка и выгрузка

Устройство, которое управляет содержанием кэша, получает требуемые данные из оперативной памяти, передает их процессору и возвращает итоги операций в оперативную память, называется кэш-контроллером.

Таким образом, главное назначение кэш-контроллера – заполнить кэш актуальными данными, причём с минимальным количеством кэш-промахов.

Загрузка кэша данными строится на принципах стратегий кэширования, а выгрузка данных — на базе 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, но и 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* (см. *Определение 2.1.12.*) или *Least Recently Replaced, LRR*);

Первый вариант самый простой и не интеллектуальный, при этом не применяется т.к. неэффективен. Алгоритмы же LRU и FIFO (см. *Определение 2.1.12.*) используются во всех современных процессорах.

Поддержание когерентности кэш-памяти

Когерентность кэш-памяти — алгоритм, который обеспечивает согласованность данных, содержащихся в оперативной и кэш-памяти, выполняется кэш-контроллером.

Если бы оперативная память (см. *Определение 1.2.9*) разрешала лишь чтение информации, то ее копия в кэш-памяти была бы идентична оригиналу в оперативной памяти. Но поскольку в оперативной памяти также можно записывать данные, обновление их в оперативной памяти очень важно для корректной работы с данными*.*

Действительно, к оперативной памяти могут обращаться не только процессор, но и периферийные устройства (см. *Определение 1.2.13.*), а в многопроцессорных и многоядерных процессорах (см. *Определение 2.2.2.*) разные процессоры или ядра. Предположим, ячейка памяти, уже преобразованная в кэше, но пока не выгруженная в основную память, получает запрос от периферийного устройства или другого процессора. Тогда кэш-контроллеру нужно прежде всего обновить данные соответствующей ячейки оперативной памяти, в противном случае считаются неактуальные данные. Точно также же, в случае изменения содержимого ячейки оперативной памяти периферийным устройством или другим процессором, кэш-контроллер обязан проверить, загружены ли обновленные данные ячейки в его кэш-память, и если нет, то это обязательно сделать.

Значит, необходимо вовремя поддерживать когерентность между кэшем и оперативной памятью.

Пути реализации когерентности:

1. Сквозная запись (*Write True*) — простейшее, но самое неэффективное решение поддержки когерентности. Оперативная память кэшируется исключительно на чтение, а запись производится, пропуская кэш, прямо в оперативную память. Исходя из своей неэффективности алгоритм не применяется в современных процессорах.

2. Алгоритм обратной записи (*Write Back*) — сложнее предыдущего. Для мониторинга изменения данных в кэш-памяти все кэш-строки содержат индивидуальный флаг (см. *Определение 2.3.5*) состояния. Если данные кэш-строки изменились, то флаг становится в состояние «модифицированное» (*Dirty*). При обращении к памяти периферийного устройства кэш-контроллер делает проверку на наличие запрашиваемых данных в кэше, и если такие есть, то по флагу состояния устанавливается актуальность данных в оперативной памяти. Если значение флага «Dirty», то в оперативной памяти данные не актуальны, и содержимое этой кэш-строки выгружается в оперативную память, а флаг ставится в состояние «не модифицировано» (*Clear*). При замещении кэш-строк кэш-контроллер таким же образом вначале анализирует состояние флага, и если строка была изменена, то она предварительно выгружается в оперативную память.

3. Протокол MESI применяется в реализации когерентности в современных процессорах. Он является модификацией алгоритма Write Back с четырьмя допустимыми значениями флага изменения кэш-строки:

* Модифицированная (*M, Modified*). Текущая строка была преобразована, но до памяти эти преобразования пока не дошли. Возможны чтение и запись в нее без опроса остальных.
* Эксклюзивная (*E, Exclusive*). Данные этой строки уникальны и не хранятся в остальных кэшах (для кэша более, чем одного уровня), кроме того они полностью совпадают с данными в оперативной памяти. Запись и чтение осуществляется без внешних запросов, т.к. она содержится только в одном кэше. После записи такая строка меняет значение флага на Modified.
* Разделяемая (*S, Shared*). Строка может находится в кэшах сразу нескольких устройств и применяться совместно. Запросы на запись в такую строку направляются на общую шину, вследствие чего все строки с таковым адресом отмечаются как старые в остальных кэшах. В это время данные основной памяти тоже обновляются. А чтение из такой строки, не запрашивает никаких внешних запросов.
* Недействительная (*I, Invalid*). Строка является недопустимой, и попытка прочитать ее содержимое повлечет кэш-промах. Данный флаг (см. *Определение 2.3.5*) устанавливается, когда строка пуста или имеет неактуальную информацию.

1.2.2. Прямой доступ к памяти

Прямой доступ к памяти (ПДП) — это главный способ разгрузить процессор от операций обмена с памятью.

Определение процессора см. в главе 1.1.

*Определение 1.2.3.*

Сопроцессор — это специализированный процессор, расширяющий возможности центрального процессора компьютерной системы, но оформленный как отдельный функциональный модуль. Физически сопроцессор может быть обособленной 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 (см. *Определение 2.3.1*). 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 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 микроконтроллер 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.

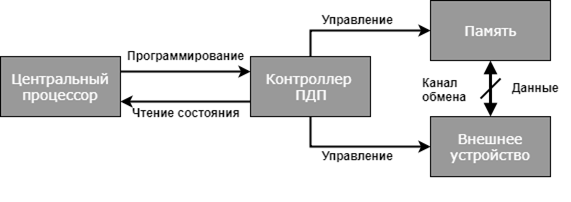
****

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

Принципы организации ПДП.

Структура координации канала ПДП показана на Рис. 1.2.2.1. Прямой доступ к памяти открывается по истечении работы данного машинного цикла [процессора](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html). Отличаясь от прерывания (см. *Определение 2.3.1*) обмен по каналу ПДП проводится без участия программы, благодаря чему данные в работающих регистрах процессора сохраняются, и вхождение в режим ПДП не нуждается в расходе времени (нет потребности хранить в стеке данные работающих регистров процессора).

Центральный [процессор](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html) осуществляет программирование микроконтроллера, налаживая его на определенный режим работы, и контролируя состояние. В процессе обмена данными по каналу ПДП процессор не активен, а микроконтроллер формирует сигналы управления обменом для памяти и внешнего устройства. Контакт внешнего устройства с памятью осуществляется по шинам адреса и данных системного интерфейса.

Сложность общего использования шин центрального процессора и внешнего устройства устраняется построением режима обмена с «захватом цикла» или (пакетного, или непрерывного) режима с блокировкой центрального процессора.

Для первого варианта решения обмен производится единичными передачами, тогда как для прямого доступа к памяти применяются отдельные циклы (такты), другими словами, передача данных (слов) чередуется с выполнением программы.

Одна из разновидностей обмена с «захватом цикла» заключается в применении таких тактов, в которых центральный процессор не меняется данными с памятью. При этом микроконтроллер ПДП должен знать эти такты. Некоторые [процессоры](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html) формируют дополнительный сигнал, определяющий обращается ли процессор в текущем цикле к памяти. Использование такого варианта построения обмена не сокращает продуктивность [процессоров](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html), но тем не менее:

● вызывает вспомогательные аппаратные затраты и допускает реализацию исключительно произвольных, нерегулярных передач;

● скорость обмена будет не быстрой, а темп нерегулярен, поскольку продолжительности циклов неодинаковых команд отличаются, к тому же, прямой доступ, возможно, отсрочит время завершения программы, если цикл ПДП не превосходит интервал, соответствующий такту процессора.

Следующая разновидность способа с «захватом цикла» встречается чаще. Здесь центральный процессор принудительно отключается от системных шин адреса и данных. Его осуществление связано с включением двух линий для передачи сигналов запроса на захват шин (ЗЗхв) и подтверждения захвата (ПЗхв). Сигнал ЗЗхв вырабатывается микроконтроллером ПДП.

После приема сигнала ЗЗхв процессор:

● прекращает на некоторое время выполнение текущей команды, не дожидаясь ее завершения;

● передает в системный интерфейс сигнал подтверждения захвата;

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

После приема сигнала ПЗхв микроконтроллер ПДП обращается к шинам системного интерфейса для обмена байтом или словом между ВУ и памятью. После этого определяет сигнал запроса на захват и передает управление шинами центральному процессору. Подготовив следующий байт или слово данных, микроконтроллер ПДП снова передает сигнал ЗЗхв [процессору](http://pue8.ru/protsessory/665-funktsii-i-sostav-protsessora-vosmi-shestnadtsatirazryadnyj-protsessor-soprotsessor.html) и т. д.

Как говорилось ранее, режим ПДП не нуждается в удержании информации о состоянии регистров процессора в стеке. Из-за этого передача данных с «захватом цикла» выполняется с более высокой скоростью, чем при обмене в режиме прерываний (см. *Определение 2.3.1*).

Для обмена данными по каналу ПДП необходима предварительная подготовка микроконтроллера, заключающаяся в установлении программы загрузки требуемых параметров для передачи:

● число передаваемых байтов (слов) данных;

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

● направление передачи (запись/чтение).

Для работы с перечисленными параметрами в микроконтроллере рассчитаны 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, используемая 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 энергонезависимой памяти может достигать более одного Тбайта (пример - жесткий диск), но при этом, как правило, значительно уступает оперативной памяти в быстродействии.

*Определение 1.2.7.*

Флэш-память — тип долговременной памяти, информацию которой можно удалить или перепрограммировать электрическим методом.

*Определение 1.2.8.*

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, обрабатываемой 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.

[Понятие о масштабируемой разрядности](#1302m92).

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.1.2.3). 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.

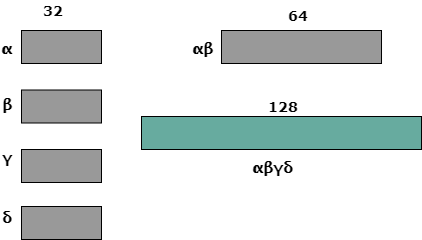
****

Рис. 1.2.3.1. Масштабируемая разрядность

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

Элементарная 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.1.2.3.3).

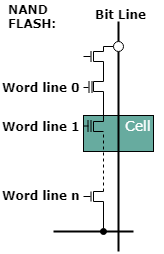
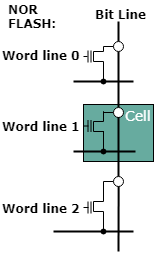
**** ****

Рис. 1.2.3.2. NAND флэш-память Рис.1.2.3.3. 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

* Количество попыток на запись в такую память ограничено (предел - миллион раз), в то время как чтение флеш-памяти бесконечно.
* Карты флеш-памяти компактны и гарантируют сохранность данных при их использовании в мобильных устройствах.
* Карты флеш-памяти обладают высокой плотностью упаковки и низким энергопотреблением.
* Флэш-память имеет особую технологию организации запоминающих ячеек. Ячейки сформированы в матрицу. Любая ячейка состоит из одного полевого транзистора. Разрядность данных внешнего интерфейса — 8 (16) бит. Стертые ячейки содержат единицу во всех битах. В процессе записи необходимые биты зануляются. Стирание реализуется для всей матрицы ячеек, иными словами, стирание одиночной ячейки невозможно, что влечет увеличение производительности в режиме записи.
* Алгоритм чтения флэш-памяти: подается адрес ячейки, затем через определенное время доступа на выходе получаем данные. Алгоритм записи данных: для программирования каждого байта приходится выполнять операции записи и считывания, которые адресованы к микросхеме флеш-памяти. Таким образом, в устройстве с флэш-памятью возможно перепрограммирование (перезапись) без извлечения микросхем из устройства.
* Время доступа флеш-памяти при чтении данных составляет 35-200 нс. Стирание информации у микросхем занимает меньше 1-2 секунд. На случай ошибочной записи (стирания) существуют методы аппаратной и программной защиты. Под программной защитой понимается последовательность команд, в случае ее нарушения операции стирания и записи не начнутся. Аппаратная защита не позволяет стирание и запись, если на входы не поданы необходимые уровни напряжения.

1.2.4. Оперативная память

*Определение 1.2.9.*

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.

Функции ОЗУ:

* Хранение, обработка данных и команд.
* Хранение модифицированных обновленных данных.
* Программирование и считывание ячеек.

Ключевые свойства ОЗУ:

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

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.

1.2.5. SRAM

Основное преимущество памяти статического типа – высокая скорость чтения и записи. Это достигается ввиду минимального времени доступа микросхемы SRAM (не больше 2 нс) и возможностью синхронно работать с процессорами. К плюсам статической памяти также следует отнести низкое энергопотребление, простое устройство интерфейса и надежность в использовании.  
Область применения — кэширование информации в процессоре и других устройствах (например, в регистрах процессора).

SRAM содержит совокупность триггеров (переключателей, которые могут находиться в одном из двух состояний: логический ноль или единица), каждый из которых содержит 1 бит данных. Как ячейки DRAM, триггеры соединяются 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 (6-8 элементов на один бит, тогда как у DRAM – 2). Таким образом, микросхемы статической памяти имеют бо́льшие габариты, из-за чего значительно возрастает стоимость за килобайт памяти.

Отмечают следующие типы статической памяти:

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

Именно на данном типе SRAM в современных процессорах строится кэш первого уровня.

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

Отличается быстрым доступом к информации.

1.2.6. DRAM

Оперативная память ПК, которая предоставляется программе в процессе ее работы, исключая сегмент данных, стек и тело программы. Динамическое размещение данных определяет применение динамической памяти в процессе работы программы. При таком размещении предварительно не известны ни тип, ни число данных, также к ним невозможно обратиться по именам, будто к статическим переменным.

DRAM обладает большой емкостью и стоимостью порядком ниже SRAM, но при этом она намного медленнее. Так как динамическая память производится на базе конденсаторов малой емкости, быстро теряющих заряд, из-за чего, чтобы не потерять хранимые данные, конденсаторы нужно подзаряжать через установленные промежутки времени. Такой процесс называется регенерацией памяти и значительно замедляет работу системы, т.к. во время него невозможно обмениваться данными с памятью. Регенерация осуществляется специальным микроконтроллером, установленным либо на кристалле центрального процессора, либо на материнской плате.

*Определение 1.2.10.*

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 1.2.11.*

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.

*Определение 1.2.12.*

Порт — это какое-либо средство ввода/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 2.2.8.)* состоит из трех частей:

* Шина адреса (см. *Определение 2.6.10*)
* Шина данных, по которой, собственно, и будет передана необходимая информация;
* Шина управления (см. *Определение 2.6.12*)

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

*Определение 1.2.13.*

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.

Потребление энергии в разных режимах работы ЭВМ

Выделяют следующие основные состояния «системы в целом»:

* Обычный – S0: Все компоненты компьютера потребляют столько энергии, сколько им нужно.
* Сон – S3: Возобновление работы с компьютером с прежнего места, происходит значительно быстрее, чем из режима гибернации (см. *Определение 1.2.13*).

В состоянии сна питание подаётся на ОЗУ и материнскую плату. Выключаются периферийные устройства, а на процессор, оперативную память, беспроводные модули и прочие компоненты в таком режиме подается минимальное напряжение. Выход компьютера из режима сна в рабочий режим практически мгновенный, где рабочий стол со всеми запущенными программами возвращается в том состоянии, в котором они пребывали перед переходом в сон. Содержимое ОЗУ полностью сохраняется, включая кэшированные данные (*Prefetch* и *Superfetch*).

Режим сна весьма удобен для повседневного использования на десктопе. SSD или HDD диск – значения не имеет, компьютер включается мгновенно. Так как в ОЗУ сохраняются кэшированные данные, то повторный запуск приложений очень быстрый относительно холодного старта компьютера, что опять же нивелирует разницу в скорости между SSD и HDD.

Такое потребление энергии очень настолько мало, что даже Wi-Fi роутер, VoIP адаптер и телевизор в режиме ожидания потребляют больше, по 4-5 Вт каждый.

* Гибернация – S4

*Определение 1.2.13.*

Гибернация — это технология, которая позволяет «выключить» компьютер, при этом не закрывая открытые приложения и не теряя несохраненные 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.

Компьютер обесточивается полностью (остается только питание на кварцевом генераторе для поддержания даты/времени), таким образом, это самый энергосберегающий режим. В нём всё содержимое оперативной памяти сохраняется в энергонезависимой памяти, такой, как жёсткий диск: состояние операционной системы, всех приложений, открытых документов и т. д. Это означает, что 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, аналогично режиму S3 (сон). Иными словами, компьютер запоминает состояние системы перед уходом в режим гибернации, а затем восстанавливает все вкладки и настройки, какие были.

Различие между S4 и S3, кроме дополнительного времени на перемещение содержимого оперативной памяти на диск и назад, в том, что перебои с питанием компьютера в S3 приведут к потере всех данных в оперативной памяти, включая все несохраненные документы, в то время как компьютер в S4 этому не подвержен.

В следующей таблице кратко описаны вышеперечисленные состояния энергопотребления.

|  |  |  |  |
| --- | --- | --- | --- |
| Состояние | Энергопотребление | Возобновление работы ПО | Аппаратная задержка |
| S0 | Максимальное | - | - |
| S1 | S1<S1≤S0 | Система возобновляет работу с прерванной точки (в S0) | <2sec |
| S2 | S3<S2≤S1 | Система возобновляет работу с прерванной точки (в S0) | ≥2sec |
| S3 | S3≤S2 | Система возобновляет работу с прерванной точки (в S0) | ≥2sec |
| S4 | Ток подается только на кнопку включения электропитания, происходит активация электроники | Система перезапускается из сохраненного файла гибернации и возобновляет работу с прерванной точки (в S0) | Длительная и неопределенная |
| S5 | Ток подается только на кнопку включения электропитания | Система загружается «с нуля» | Длительная и неопределенная |

Таблица 1.2.7.1. Состояния энергопотребления

Охлаждение

* Система воздушного охлаждения
* 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: экономия, надежная работа, безопасность, отсутствие шума.

Недостатки: низкая эффективность для современного оборудования.

* Активная

Составляющие системы: вентилятор, радиатор, который очень часто для эффективности работы добавляется.

Принцип работы:горячий воздух отводится за пределы 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.

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

*Определение 1.3.1.*

MISD (англ. *Multiple Instruction stream, Single Data stream*) — тип архитектуры параллельных вычислений (см. *Определение 4.1.1*), в котором несколько функциональных модулей (два или более) совершают разные операции над одними данными.

*Определение 1.3.2.*

MIMD (англ. *Multiple Instruction stream, Multiple Data stream*) — многовариантный поток команд и данных, который включает некоторое число независимых процессоров, асинхронно выполняющих различные команды над различными данными. 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 1.3.3.*

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 1.3.4.*

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

*Определение 1.3.5.*

SSE (*Streaming SIMD Extensions*) *—* ранняя версия AVX, которая 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 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.

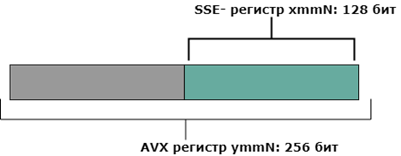
****

Рис. 1.3.1. AVX и SSE регистры

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

Улучшения AVX:

* 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 или два независимо работающих числа, каждое из которых 256-разрядное. Помножив два таких регистра, получится заготовка для скалярного произведения. Потом можно разделить пополам и получить четыре 128-разрядных числа; и так далее вплоть до char (8 бит). Тогда результатом будет 64 числа — столько элементов размером 1 char можно хранить в одном регистре, который будет работать как вектор. Операция, совершаемая над этим регистром, выполняется независимо над каждым кусочком в один машинный такт. Это значительно повышает производительность для интенсивных вычислений с плавающей точкой.  
AVX специализируется на работе с числами с плавающей точкой типа float и double.

*Определение 1.3.6.*

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 наибольшее 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 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 микроконтроллеров 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 и хранения данных, автоматически решают вопрос, какие данные использовать. Google позволяет через интернет обучать нейросети, используя их удаленные процессоры.

Архитектура систем сотовых связей тоже есть гибкая архитектура. Система сотовой связи допускает большое разнообразие вариантов конфигураций 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».

Динамическое балансирование используется там, где ответственность за ошибку ниже: 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.

Статическое распараллеливание используется при жесткой системе, т.е. вероятность ошибки должна стремиться к нулю. Кроме того, при таком решении задачи центральный компьютер может считать вероятность того, что найденное решение правильное.

## 1.4. Специализированные вычислители.

*Определение 1.4.1.*

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 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 быть отнесены к одному из трех типов:

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

*Определение 1.4.2.*

Специализированный процессор — 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. Они автоматически защищают информацию, т.е. её запаковывают и распаковывают из защищенного вида.

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 (см. *Определение 2.4.6*), 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 микроконтроллеры освобождают компьютерные центральные процессоры от самых медленных операций ввода или вывода информации.

*Определение 1.4.3.*

Математический сопроцессор (англ. *floating point unit, FPU*, сопроцессор с плавающей точкой) — это сопроцессор, расширяющий командное множество центрального процессора и обеспечивающий 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.

*Определение 1.4.4.*

Модуль операций с плавающей точкой — это часть процессора, используемая 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 1.4.5.*

Цифровой сигнальный процессор (ЦСП) — процессоры, оптимизированные для решения задач обработки электрических (радио, видео, аудио и других) сигналов в цифровой форме, как правило, в режиме реального времени.

Примеры использования ЦСП:

* Музыкальные и речевые синтезаторы;
* Кодирование аудио (видео) потоков;
* Распознавание речи, изображений;
* Управление технологическими процессами.

Математически эти задачи сводятся к поэлементному перемножению элементов многомерных векторов действительных чисел и к последующему суммированию этих 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 1.4.6.*

Потоковые процессоры — процессоры, в основе которых лежит принцип обработки многих данных с помощью одной команды. Эта технология позволяет выполнить одно и то же действие, например, вычитание 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 1.4.7.*

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 позволяет обрабатывать визуальную информацию эффективнее, чем центральный CPU компьютера.

*Определение 1.4.8.*

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

*Определение 1.4.9.*

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.

## 1.5. Графический процессор

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

У GPU нет инструментов прямого взаимодействия с оперативной памятью компьютера и с устройствами ввода/вывода (помимо монитора). По этой причине руководство графическим процессором проводится исключительно посредством центрального процессора. Модель взаимодействия центрального и графического процессоров представлено на рис. 1.5.1.

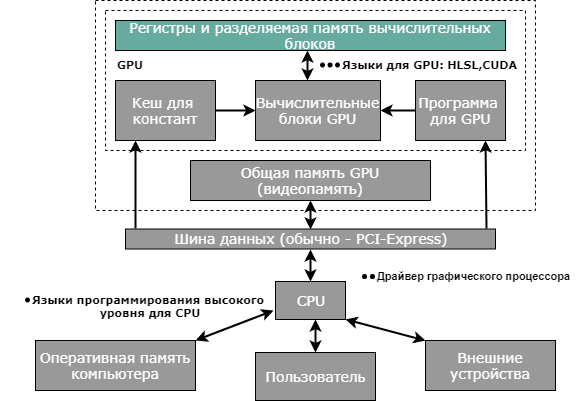
****

Рис. 1.5.1. Модель взаимодействия центрального и графического процессоров

Графические ускорители присоединяются к системной плате ПК посредством высокоскоростной шины данных (в настоящее время PCI-Express). С помощью неё CPU получает доступ к видеопамяти, а также к определенным разделам кэш-памяти, находящейся на графическом процессоре. Посредством шины центральный процессор загружает в графический процессор программу и запускает её.

Прежде чем запустить программу, исполняемую на GPU, центральный процессор посылает графическому следующие данные:

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

Так как к константам нужен регулярный и быстрый доступ, то они вносятся в кэш-память (или регистры), размещенную на кристалле GPU. Массивы данных нередко могут быть столь огромны, что полностью в кэш-память не помещаются, но при элементарной потоковой обработке все элементы массивов данных применяются единственный раз. Таким образом, для хранения этих массивов предназначена видеопамять (общая память), которая представляет собой отдельные микросхемы на плате графического ускорителя, функционирует медленнее кэш-памяти и регистров, однако обладает бόльшим объемом .

Результаты своей работы GPU тут же заносит в область видеопамяти, которая именуется буфером кадра, посылающим их на монитор. Однако имеется допустимость не показывать результаты на экране, а дублировать их из видео-памяти в оперативную память компьютера, откуда они оказываются доступными для последующей обработки центральным процессором. Это основополагающее применение графических процессоров для вычислений общего направления, не связанных с обработкой графики.

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, это значит, что 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:

* Встроенный. Он есть практически во всех компьютерах. Его устанавливают в CPU, чтобы сделать потребление энергии в несколько раз ниже.
* Дискретный. Его устанавливают на отдельные модули, так как он отличается своей мощью, но ему необходимо отличное охлаждение.

## 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.

Существует BIOS 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 (см. *Определение 2.3.3*). 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 кучу других функций), однако могут работать быстрее и надежней при обстоятельствах, в которых ОС общего назначения затормозит. Подобный переход от ОСРВ к RTE представлен в таблице 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. |

Таблица 1.6.1. Переход от ОСРВ к RTE системе.

# Глава 2. Процессор (компьютер)

## 2.1. Узлы процессора

*Определение 2.1.1.*

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 (ЦП) *—* это выполнение программ, которые находятся в основной памяти.

*\*определение регистра см. определение 1.1.8.*

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 (см. *Определение 2.6.10*).

*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.*

Регистр указателя стека *—* это 16-разрядныйрегистр, 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 (см. *Определение 2.1.8.*) 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 | Выбор банка регистров. Устанавливается и сбрасывается  программой для выбора рабочего банка регистров |
| RS1 | PSW.4 |
| 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 2.1.2. Формат слова состояния программы (PSW)

Флаг (см. *Определение 2.3.5*) 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 2.1.8.*

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

Большинство компьютеров имеет как внутренние (см. *Определение 1.1.9*), так и внешние шины (см. *Определение 1.1.10*).

*Определение 2.1.9.*

Внутренняя шина (локальная) — шина, с помощью которой все внутренние компоненты компьютера подключаются к материнской плате.

К важнейшим внутренним шинам относятся:

* системная шина(см. *Определение 2.2.8.*) 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.

*Определение 2.1.10.*

Внешняя шина — это шина, подключающая внешнюю периферию (см.ниже) к материнской плате.

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;
* последовательные порты СОМ;
* параллельные порты LPT;
* последовательный порт FireWire;
* сетевой порт RJ45;
* порт модема RJ11 для телефонной линии;
* порты аудиоинтерфейса.

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 задающих и подчиненных устройств 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.):

* микропрограммное устройство, которое задает последовательность микрокоманд;
* операционное устройство, реализующее заданную последовательность микрокоманд.

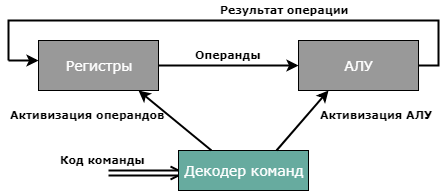
****

Рис 2.1.1. 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.

Счетчик команд работает с памятью, которая находится внутри процессора. Эта память носит название очередь команд (см. *Определение 2.1.11.*), 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 – ряд операций, совершающихся процессором при выполнении одной машинной команды. Вместе с этим процессор должен произвести выборку, раскодирование и выполнение*.* Когда команда содержит операнд, находящийся в оперативной памяти, то процессору придется провести дополнительно две операции: выборку операнда из памяти и запись вычислений в память. Ниже охарактеризованы каждое из перечисленных действий.

* Выборка команды. Блок управления достает команду из памяти (из очереди команд), переписывает ее во внутреннюю память процессора и повышает значение счетчика команд на длину этой команды.
* Раскодирование команды*.* Блок управления распознает тип выполняемой команды, переправляет заданные в ней операнды в АЛУ и вырабатывает электрические сигналы управления АЛУ, соответствующие типу проводимой операции.
* Выборка операндов*.* Если в команде используется операнд, расположенный в оперативной памяти, то блок управления приступает к его выборке из памяти.
* Запись результата в память. Если результат выполнения команды должен быть сохранен в памяти, блок управления начинает операцию сохранения данных в памяти.
* Выполнение команды*.* АЛУ применяет заданную в команде операцию, сохраняет итоги вычислений и модифицирует состояние флагов (см. *Определение 2.3.5*), относительно которых программа делает выводы об итоге выполнения команды.
* Запись результата в память. В случае сохранения результата команды в памяти блок управления приступает к операции сохранения данных.

Обобщая сказанное, составим цикл выполнения команды:

1. Выбрать из очереди команд команду, на которую указывает счетчик команд.
2. Определить адрес следующей команды в очереди и записать ее адрес в счетчик команд.
3. Декодировать команду.
4. Если в команде есть операнды, находящиеся в памяти, то выбрать операнды.
5. Выполнить команду и установить флаги (см. *Определение 2.3.5*).
6. Записать результат в память (по необходимости).
7. Начать выполнение следующей команды с п.1.

*Определение 2.1.11.*

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

*Определение 2.1.12.*

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 2.1.13.*

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 2.1.14.*

Контроллер прямого доступа к памяти *—* (*англ. Direct Memory Access, DMA*) *—* блок, который позволяет некоторым аппаратным подсистемам получать доступ к основной системной памяти (оперативной памяти) независимо от 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 (см. *Определение 2.3.1*);
* 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, – 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 (см. *Определение 2.5.2*). 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
* FPU
* Блок сохранения результатов обеспечивает запись результата выполнения инструкции в ОЗУ по адресу, указанному в обрабатываемой инструкции.
* Блок работы с прерываниями (см. *Определение 2.3.1*).

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 (см. *Определение 2.3.2.*). 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 2.2.2.*

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 2.2.3.*

Многопоточность (англ. *Multithreading*) — 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 2.2.4.*

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 2.2.5.*

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 (см. *Определение 2.3.1*), 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 2.2.6.*

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, по сути, 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 необходимо 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 2.2.7.*

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

*Определение 2.2.8.*

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 2.2.9.*

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 2.2.10.*

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 (см. Список сокращений).

*Определение 2.2.11.*

«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 (см. *Определение 2.5.2*).

## 2.3 Прерывания

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.

*Определение 2.3.1*

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 (см. *Определение 2.3.2.*), который отвечает на событие и обслуживает его, после чего возвращает управление в прерванный код.

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.

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 (см. *Определение 2.4.6*), 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 случай 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.

*Определение 2.3.2.*

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.

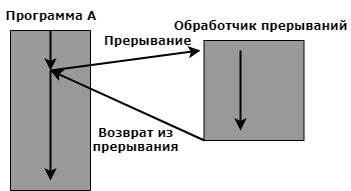
****

Рис. 2.3.1. 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. Такой адрес состоит из пары 2-байтовых слов, поэтому каждый из векторов занимает четыре байта.

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

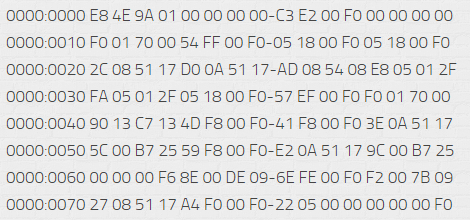
****

Рис. 2.3.2. Таблица векторов прерываний в компьютере

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:

* Распознавание или классификация прерываний.
* Передача управления соответственно обработчику прерываний.
* Правильное возвращение к прерванной программе.

Типы прерываний

Прерывания, возникающие при работе вычислительной системы, как мы писали выше можно разделить на 4 группы:

****

Рис. 2.3.3. Типы прерываний

*Определение 2.3.3*

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.

*Определение 2.3.4*

Внешние прерывания возникают по сигналу внешних устройств и подразделяются на два вида: немаскируемые и маскируемые.

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

* Маскируемые прерывания могут быть замаскированы программными средствами компьютера;
* 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. По 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.

*Определение 2.3.5*

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 2.3.4. 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 (см. *Определение 3.3.4*).

*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 состоянии 0 строки будут обрабатываться в сторону увеличения адресов, при 1 – 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 оно 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.

## 2.4. Представление о микросхемах обвязки процессоров

*Определение 2.4.1*

Чипсет (англ. *chipset*) — набор микросхем, которые обычно применяются при создании системных плат для «обвязки» процессора, спроектированных для совместной работы с целью выполнения набора каких-либо функций.

Чипсет в компьютере, размещаемый на материнской плате, выполняет роль связующего компонента, обеспечивающего совместное функционирование подсистем памяти, центрального процессора, устройств ввода/вывода и других. Чипсеты встречаются и в других устройствах, например, в сотовых телефонах.

*Определение 2.4.2*

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, видеокарта и несколько микроконтроллеров шин. Всё чаще для современных процессоров требуется все меньше и меньше обвязки. Но при этом, процессору по-прежнему нужны шины, внешние устройства, которые в большинстве случаев удается уместить в корпус процессора. Конечно же, с точки зрения архитектуры, все это отдельные части, которые просто находятся под одной крышкой.

*Определение 2.4.3.*

Аппаратный порт — разъем специального назначения в компьютере, предназначенный для предоставлении подключения оборудования какого-либо установленного типа.

Как правило, портами называют разъемы, необходимые для работы периферийного оборудования, значительно обособленного от архитектуры компьютера.

Северный и южный мосты компьютера (вернее материнской платы) — два основополагающих функциональных микроконтроллера, отвечающих за деятельность всех элементов системной платы и называются чипсетом.

*Определение 2.4.4.*

Северный мост (англ. *Northbridge*) — системный микроконтроллер, считающийся одним из компонентов чипсета материнской платы, отвечающий за взаимодействие с оперативной памятью (RAM), видеоадаптером и процессором (CPU).

Ответственен за частоту системной шины, тип оперативной памяти и ее предельно допустимый объем. Одна из центральных функций северного моста *—* обеспечение работы системной платы и процессора, и вместе с этим установление скорости работы. Во многие современные материнские платы по умолчанию встраивается видеоадаптер. Также функциональная черта северного моста заключается в управлении шиной видеоадаптера и ее быстродействием. К тому же северный мост отвечает за связь всех вышеперечисленных устройств с южным мостом.

*Определение 2.4.5*

Южный мост (англ. *Southbridge*) — функциональный микроконтроллер, чаще называемый контроллером ввода-вывода или ICH (*In/Out Controller Hub*). Контролирует »медленные» операции, к числу которых принадлежит обработка взаимодействия между интерфейсами IDE, SATA, USB, LAN, Embeded Audio и северным мостом системы, который напрямую соединен с процессором и другими значимыми составляющими (оперативная память, видеоподсистема и т.п.). Также южный мост ответственен за обработку данных на шинах PCI, PCIe.

*Определение 2.4.6*

Таймер — многофункциональное устройство, задающее временный интервал. Представляет собой ступень между аналоговыми и цифровыми устройствами. С помощью него осуществляется конвертирование аналоговых сигналов в последовательность импульсов по заданному закону.

*Определение 2.4.7*

Контроллер прерываний (англ. *Programmable Interrupt Controller, PIC*) — микросхема или встроенный узел процессора, определяющий возможность ступенчатой обработки запросов на прерывание от других устройств.

*Определение 2.4.8*

Цифро-аналоговый преобразователь (ЦАП) — устройство, выполняющее преобразование цифрового (обычно двоичного) кода в аналоговый сигнал (ток, напряжение или заряд), служит для взаимодействия дискретного цифрового мира и аналоговых сигналов.

Аналого-цифровой преобразователь (АЦП) осуществляет обратную операцию.

## 2.5. 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.

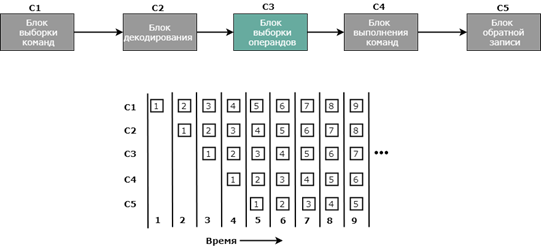
****

Рис. 2.5.1 – 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 2.5.1*

RISC (англ. *restricted (reduced) instruction set computer* — «компьютер с сокращенным набором команд») — архитектура процессора, при которой 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 (см. *Определение 2.5.2*). 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, сами по себе они усложняют схему декодирования номера регистра, тем самым увеличивая время доступа к регистрам.

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

1. Процессор с расширенным набором команд

*Определение 2.5.2*

CISC (англ. *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.

И все же, CISC-подход привел к невыполнению отдельных команд чисто аппаратными средствами (при целесообразной сложности таких средств). В итоге в процессорах возникли блоки, «с ходу» замещающие некоторые команды, в особенности сложные, последовательностями из более простых команд. К тому же, опыты дали понять, что многие сложные команды при разработке программ становились не нужны. В конечном счете, по причине высокой сложности команд и их большого количества устройство управления ВМ приходилось строить исключительно на базе программируемой логики, а именно с использованием «медленной» управляющей памяти. Последний фактор значительно снижает возможность увеличения тактовой частоты процессора. Все эти условия дали поворот в сторону RISC-архитектуры. Между тем целый ряд бесспорных преимуществ CISC-архитектуры поддерживают ее актуальность (в первую очередь для разработчиков программных приложений). В связи с этим ведущие фирмы-производители ВМ (Intel, AMD, IBM и др.) в собственных новейших разработках, до сих пор, не отказываются от CISC-подхода.

1. Процессор со сверхдлинными командами

*Определение 2.5.3*

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 (см. *Определение 3.3.2*) формирует одно «очень длинное командное слово».

Идея VLIW базируется на том, что задача эффективного планирования параллельного выполнения команд возлагается на «разумный» компилятор (см. *Определение 3.2.1*). 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, 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 для VLIW-процессоров программы 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 2.5.4*

SISD (*Single Instruction stream / 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 (см. *Определение 2.5.5*), конвейерные и VLIW–процессоры.

SIMD

В SIMD–компьютеры входит один командный процессор (управляющий модуль), который называется микроконтроллером, и процессорные элементы (модули обработки данных). Управляющий модуль принимает, анализирует и выполняет команды. Если в команде встречаются данные, микроконтроллер 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 логика вычислений реализована более эффективно. Почти половина логических инструкций обычного процессора связана с управлением выполнения машинных команд, а остальная часть относится к выполнению арифметических операций и работе с внутренней памятью процессора.

Архитектура ILP–процессоров

Увеличение производительности достигается благодаря модернизации полупроводниковой технологии и повышению плотности микросхем. Использование реализации установленного типа параллелизма тоже помогает добиться роста скорости выполнения программ. Параллелизм на уровне команд (instruction-level parallelism, ILP) оказался возможен за счет построения методик компиляции (см. *Определение 3.2.3*) и процессоров, которые благодаря параллельному выполнению одиночных RISC-операций ускоряют работу. Системы на основе ILP применяют программы, составленные на традиционных языках высокого уровня для последовательных процессоров, а выявление «скрытого параллелизма» автоматически проводится с использованием соответствующей компиляторной технологии и аппаратного обеспечения.

Суперскалярные процессоры

*Определение 2.5.5*

Суперскалярные процессоры — реализация ILP-процессора для последовательных архитектур, программа для которых не должна и не имеет возможности предоставлять точных сведений о параллелизме. Так как программа обладает точными сведениями о наличии ILP, задача выявления параллелизма решается аппаратурой, формирующей план действий для выявления «скрытого параллелизма». Код для суперскалярных процессоров включает цепочку инструкций дающую корректный результат, если совершается в указанном порядке. Программа задает последовательный алгоритм, не зная точную природу аппаратного обеспечения, на котором будет выполнять работу или конкретный временной порядок, в котором будут осуществляться команды. Данный подход повышает сложность аппаратного обеспечения. Кроме того, суперскалярный процессор формирует план выполнения, применяя преимущества факторов, определяющихся непосредственно во время выполнения.

Разрешенные для изменения области динамической структуры программы указывают существующие на множестве инструкций связи: зависимость по данным и по управлению. При описании архитектур суперскалярных процессоров чаще всего используется модель окна исполнения. При исполнении программы микропроцессор как бы продвигает по статической структуре программы окно исполнения. Если между командами нет зависимости, то они могут совершаться в окне параллельно.

*Определение 2.5.6*

Тип архитектуры EPIC (*EPIC, Explicitly Parallel Instruction Computing* — «вычисление с явным параллелизмом машинных команд») — развитие архитектуры VLIW, собравшее в себе многие понятия суперскалярной архитектуры, но адаптированные к архитектуре EPIC.

В сущности, EPIC является «идеологией», которая определяет, как инициировать ILP-процессоры, и, к тому же совокупность характеристик архитектуры, поддерживающей данную идеологию множества разных архитектур наборов команд (ISA, см. Список сокращений).  
 К архитектуре EPIC можно определить для RISC. Кроме включения или исключения каких-либо параметров архитектуры, проектировщики процессоров должны учитывать требования по набору кодов операций, пределу поддерживаемых типов данных и числу применяемых регистров. Поэтому тип архитектуры EPIC должен быть архитектурой общего назначения, которая способна достигать высокой степени параллелизма на уровне команд в числовых и скалярных приложениях.

Одна из задач архитектуры EPIC — сохранение построенного в VLIW принципа статического создания плана выполнения, при этом, расширяя его возможностями суперскалярного процессора, которые помогут новой архитектуре принимать во внимание динамические факторы, традиционно ограничивающие параллелизм. Исходя из этого, «идеология» EPIC реализовывалась на таких базовых принципах, как:

* компилятор *—* основополагающий элемент в создании плана выполнения, а архитектура предусматривает поддержку компонентов;
* архитектура предоставляет функции, которые помогают компилятору использовать статические ILP;
* архитектура регламентирует механизм для передачи плана выполнения компилятора аппаратному обеспечению.

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

*Определение 2.6.1*

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 2.6.2*

Параллельная передача — синхронная передача соответствующих компонентов сигнала по двум или большее числу путей. Применяя большое количество электрических проводов допускается транслировать сразу несколько бит, что позволяет значительно увеличить скорость передачи, в отличие от последовательной передачи. Данная технология используется внутри компьютера (например, во внутренних шинах данных), в некоторых случаях и во внешних устройствах (например, в принтерах). Главная трудность — «перекос», т.к. провода при параллельной передаче содержат не полностью одинаковые свойства, следовательно, некоторые биты могут быть переданы раньше остальных, а это может исказить сообщение. Бит четности может позволить снизить число ошибок. Несмотря на это, электрический провод при параллельной передаче данных не так надежен на больших расстояниях, так как передача нарушается с очень большой вероятностью.

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

Последовательные шины 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).

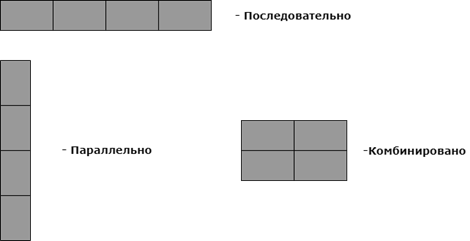
Комбинированная шина:

В каких-то случаях выгодно огромный кусок данных разбить так, чтобы некоторые его части передавались 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. 2.6.1. Варианты передачи 4 элементов данных

Классификация и особенности шин в отношении метода синхронизации:

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

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 2.6.3*

Синхронный метод последовательной передачи данныхпозволяет передавать данные блоками. Шаблонная схема передачи данных при синхронном методе: в процессе сопоставления работы приемника и передатчика в начале блока посылают биты синхронизации; затем передаются данные, символ, означающий окончание передачи, и код обнаружения ошибки. При синхронной передаче данные могут посылаться как в виде символов, так и потоком битов. Под кодом обнаружения ошибки понимается циклический избыточный код обнаружения ошибок, определяющийся по содержимому поля, благодаря которому определяют достоверность информации.

*Определение 2.6.4*

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.

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 (см. Список сокращений) 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 отправкой 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.

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 2.6.5*

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 2.6.6*

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 2.6.7*

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, перестанет 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 2.6.8*

Комбинированными топологияминазываются топологии, которые комбинируют компоновку сети по принципу шины, звезды и кольца. Звезда-шина *(star-bus)* — это комбинация топологий «звезда» и «шина». Если несколько 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 2.6.9*

Звезда-кольцо *(star-ring)* немного похожа на звезду-шину. В обеих топологиях 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.

|  |  |  |
| --- | --- | --- |
| Топология | Преимущества | Недостатки |
| Шина | Экономный расход кабеля. Сравнительно недорогая и несложная в использовании среда передачи. Простота, надежность. Легко расширяется. | При значительных объемах трафика 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 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. |

Таблица. 2.6.3

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 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, только 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 двух машинах, то полоса пропускания 38 оставшихся подсетей не используется.

Классификация и особенности шин в отношении характера передаваемой информации

*Определение 2.6.10*

Шина адреса — компьютерная шина (серия, линия, соединяющая два или более устройств), используемая для указания адреса ячейки памяти или устройства, с которым будет происходить обмен информацией. 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.

*Определение 2.6.11*

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

*Определение 2.6.12*

Компьютерная шина, по которой передаются сигналы, определяющие характер обмена информацией по магистрали, называется 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.

*Определение 2.6.13*

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, т.е. мультиплицируется во временной области, с дополнительными линиями управления, которые используются для последовательной передачи.

## 2.7. Виртуальные машины. Виртуализация и эмуляция. Гипервизор и супервизор. Контейнеры и докеры.

*Определение 2.7.1*

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 2.7.2*

Гостевая система — это виртуальная машина, которая использует общие средства, предоставленные хостом. Она охватывает приложение и все, что необходимо для его запуска: библиотеки и системные исполняемые файлы. Такая система несет в себе целый аппаратный стек, в том числе виртуальные сетевые адаптеры, файловое хранилище и ЦП, а также свою полноценную гостевую ОС.

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

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

Для перевода процессора в режим виртуализации, платформе виртуализации нужно вызвать инструкцию VMXON и передать управление гипервизору, который запускает виртуальную гостевую систему инструкцией VMLAUNCH и VMRESUME (точки входа в виртуальную машину), см. рис 2.7.1.

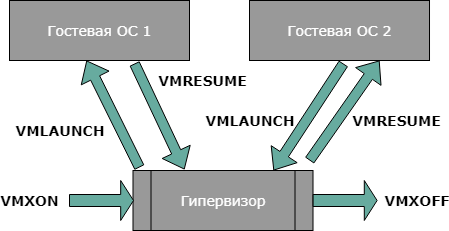
****

Рис.2.7.1. Перевод в режим виртуализации

В случае использования вызова инструкции VMXOFF, Virtual Machine Monitor сможет 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 (см. *Определение 2.7.4*). 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 фактически не насчитывает ограничений в плане архитектуры процессора;
* допустимость увеличения или уменьшения системных ресурсов: оперативной памяти, дискового пространства, вычислительных ресурсов процессора;
* так как на физическом сервере привлекается 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 эмуляции работают как интерпретаторы (см. *Определение 3.2.2*): последовательно выбирают код гостевой ОС и эмулируют разные аппаратные архитектуры. Минус подхода — высокая потеря производительности гостевой ОС.

* Частичная эмуляция (нативная виртуализация)

Для изолированного запуска виртуальная машина виртуализирует аппаратное обеспечение, что разрешает запускать гостевые ОС, разработанные только для той же архитектуры, что и у хоста. Из этого следует что, гостевые системы могут быть запущены параллельно. Данный способ виртуализации увеличивает быстродействие гостевых систем по сравнению с полной эмуляцией. Для ускорения работы, в платформах виртуализации, используется «прослойка» между гостевой ОС и оборудованием — гипервизор, который помогает гостевой системе напрямую обращаться к возможностям аппаратного обеспечения. Минус этого способа — то, что виртуальные машины зависят от архитектуры аппаратной платформы.

* Частичная виртуализация (виртуализация адресного пространства)

Виртуальная машина симулирует некоторое количество экземпляров аппаратного окружения, в частности, пространства адресов. В связи с этим можно одновременно использовать ресурсы и изолировать процессы, не разделяя экземпляры гостевых ОС. То есть, пользователем не создаются виртуальные машины, а происходит изоляция некоторых процессов на уровне ОС.

Гипервизор и супервизор.

*Определение 2.7.3*

Супервизор — программа, которая управляет процессами, значительно упрощая работу с памятью и оборудованием ОС.

*Определение 2.7.4*

Гипервизор — программа, с помощью которой можно одновременно управлять несколькими операционными системами на одном и том же хост-компьютере.

Обязанности гипервизора:

* изолирование ОС друг от друга;
* разделение ресурсов между ОС;
* максимально эффективное управление ресурсами;
* обеспечение защиты и безопасности ОС.

Контейнер, докер

Контейнеры и виртуальные машины близки относительно своих задач: изолирование приложения и его зависимостей в самостоятельный блок, который можно запускать с разных хост-компьютеров, т.к. в процессе перемещения между ними содержимое не меняется.

Главное отличие между контейнерами и виртуальными машинами — в их архитектурном подходе.

На рис.2.7.2 представлена архитектура виртуальной машины, а на рис.2.7.3 архитектура контейнера.

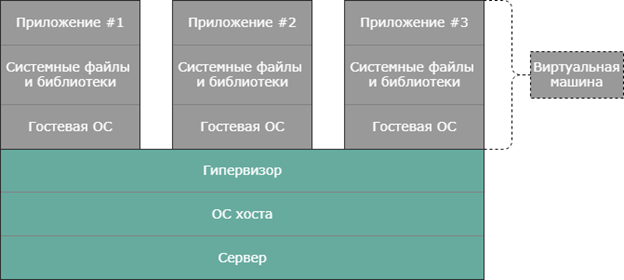
****

Рис. 2.7.2. Архитектура виртуальной машины

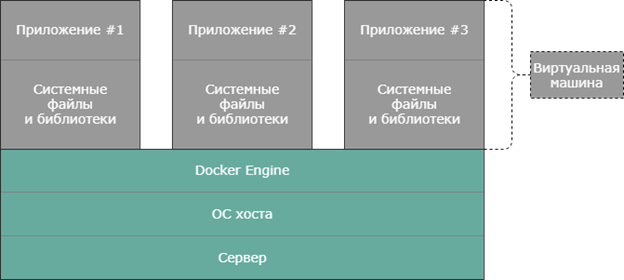
****

Рис. 2.7.3. Архитектура контейнера

Контейнеры, в отличие от виртуальной машины, которая обеспечивает аппаратную виртуализацию, обеспечивают виртуализацию ОС.

Если запуск выполняется в виде контейнера, то приложение попадает в ту же ОС, где происходит взаимодействие с остальными приложениями. Это происходит потому, что контейнеры разделяют ядро хоста с другими контейнерами. А виртуальная машина может выбрать любую доступную ей ОС. Таким образом, при запуске запросы со стороны контактирующих приложений совершают длинный путь из одной операционной среды в другую и обратно.

Еще одно различие состоит в порядке создания каждого из типов.   
Контейнеры не упаковывают ядро или виртуальную аппаратуру, а включают лишь пользовательское пространство, в которое потом можно добавить то, что еще потребуется. При этом каждый контейнер получает персональное изолированное пользовательское пространство для обеспечения возможности запуска нескольких контейнеров на одном хосте. Контейнеры разделяют между собой ресурсы хоста-контейнера, именно поэтому контейнеры настолько легковесны.  
Для виртуальных машин, наоборот, сначала берется полная операционная система, а затем, исключается то, что не понадобится для работы данного приложения.

*Определение 2.7.5*

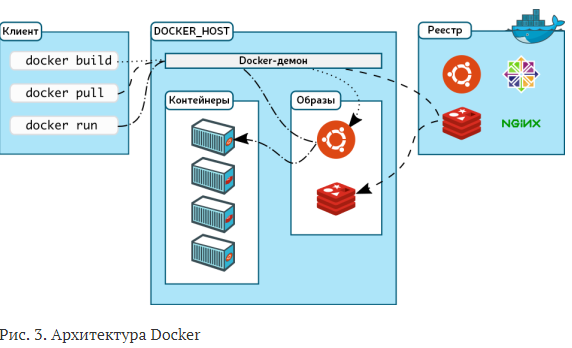
Docker — инструмент для развертывания (осуществление действий, подготавливающих программную систему к использованию) и управления приложениями в среде виртуализации на уровне операционной системы.

Главными преимуществами Docker являются:

* Легкость использования. Созданная на контейнерах docker платформа дает возможность портировать полезную нагрузку. К тому же docker-контейнеры могут работать не только на реальной локальной машине в центре обработки данных, но и на виртуальной машине в облаке.
* Скорость работы. Docker-контейнеры легковесны и быстры. Можно создать и запустить Docker-контейнер за секунды, в отличие от виртуальных машин, которые каждый раз запускают полноценную виртуальную ОС.
* Модульность и масштабируемость. Docker предоставляет возможность разделять функциональность приложения в отдельные контейнеры. Вследствие чего обновление компонентов приложения может происходить независимо друг от друга.

На рисунке 2.7.4 представлена архитектура Docker.

* Docker Daemon (демон Докера) — фоновый сервис, запускается на хост-машине, с которой взаимодействует клиент, отвечает за создание, запуск и уничтожение docker-контейнеров.
* Docker Client (клиент Докера) — программа, которая получает инструкции от пользователя и взаимодействует с docker-демоном.

**** Рис. 2.7.4 Архитектура Docker

## 2.8. 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.

Функциональное назначение ядра:

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

Варианты реализации

* Монолитное ядро. Широкий набор служебных процедур работает в привилегированном 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. Предоставляет некоторый механизм как динамической, так и статической подгрузки модулей ядра, 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 2.8.1.*

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

*Определение 2.8.2.*

Невытесняющая планировка — система планирования процессов, при которой активный процесс продолжает работать, пока по собственной инициативе, не передаст управление планировщику ОС, чтобы тот выбрал из очереди другой, готовый к реализации процесс.

* FIFO процесс, готовый к выполнению, встает в очередь готовых процессов. При прекращении выполнения текущего процесса для следующего выполнения отбирается процесс, находившийся в очереди дольше остальных.
* EDF (алгоритм с динамическим планированием задач). Здесь приоритет тем выше, чем меньше срок выполнения. В каждый цикл задачи выстраиваются в последовательности заново в зависимости от критического срока выполнения.

*Определение 2.8.3.*

Вытесняющая планировка — система, отличающаяся от невытесняющей планировки тем, что решение о переведении процессора с выполнения одного процесса на выполнение другого определяется планировщиком операционной системы, а не самой активной задачей.

* RMS (алгоритм с фиксированным приоритетом). Приоритет задачи присваивается по принципу: чем реже вызывается задача, тем выше ее приоритет.
* LSTF (алгоритм планирования). Приоритет задачи определяется в соответствии с временем связывания задачи: чем меньше, тем больше приоритет.

Механизмы переключения задач

В многозадачной системе переключение задач есть сохранение состояния задачи на момент ее останова (см. *Определение 3.3.9*). Контекстом называют информацию о задаче для последующего восстановления прерванного процесса. В системе отводится область оперативной памяти, доступная только ОС, хранящая контексты задач.

В целях минимизации времени переключения необходимо восстанавливать и сохранять минимальное количество информации о каждой задаче. Следует отметить, что задачи не являются рентабельными (не имеет возможности вызова самой себя), так как контексты сохраняются не в стеке, а в фиксированной области памяти в специальной структуре данных — сегменте состояния задачи (*Task State Segment, TSS*). Сегмент TSS определяется дескриптором, хранящимся только в GDT*(Global Descriptor Table*).

Под термином «старая задача»будем понимать ту задачу, выполнение которой прекращается. Под термином«новая задача» будем понимать ту задачу, которую начинает выполнять процессор.

Процедура, которая обращается к дескриптору TSS, может вызвать переключение задачи. Переключение задачи может повлечь следующие события:

* старая задача выполняет команду *FAR CALL (*обеспечивает перенаправление всех указывающих на него вызовов) или *FAR JMP (*обеспечивает перенаправление всех указывающих на него переходов)*,* селектор выбирает шлюз задачи;
* старая задача выполняет команду *FAR CALL* или *FAR JMP*, селектор вызывает дескриптор TSS;
* старая задача вызывает команду *IRET* для возврата в предыдущую команду (эта команда приводит к переключению задачи, если в регистре *EFLAGS* бит вложенной задачи NT =1);
* возникло аппаратное или программное прерывание и соответствующий элемент дескрипторной таблицы содержит шлюз задачи*.*

Таким образом, возможны две модели переключения задач:

* Прямое переключение задач;
* Косвенное переключение задач (переключение задач с использованием шлюзов).

Принцип работы механизма переключения задач см. рис. 2.8.1.

Процесс выполнения переключения задачи можно представить следующим образом:

Имеется TR *(task register) с* регистром дескриптора TSS, определяющий TSS старой задачи. Если селектор в командах FAR JMP, FAR CALL, IRET(NT = 1), INT указывает прямо (дескриптор TSS) или косвенно (шлюз задачи) в GDT на системный объект переключения задачи, то производится переключение задачи:

* процессор сохраняет контекст старой задачи в сегменте TSS старой задачи;
* процессор загружает в TR селектор сегмента новой задачи;
* процессор загружает в сегмент TSS новой задачи селектор TSS старой задачи (в поле обратной связи);
* получив доступ к сегменту TSS новой задачи, процессор загружает текст новой задачи в регистры.
* процессор устанавливает флаги NT (в регистре EFLAGS) и TS, устанавливает бит занятости задачи в дескрипторе TSS новой задачи.

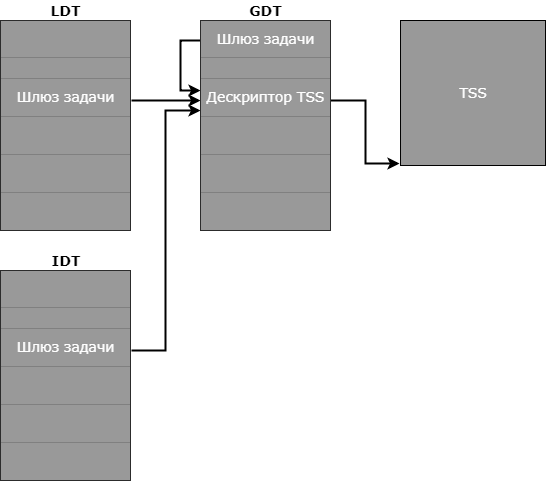
****

Рис. 2.8.1. Принцип работы механизма переключения задач

# Глава 3. Процессор (программирование)

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

*Определение 3.1.2*

Распределённые вычисления — метод построения компьютерных вычислений, отличающийся от параллельных тем, что вычислительные процессы выполняются не на одном компьютере, а распределяются по разным, но при этом показатели работы такой системы активно изменяются и не всегда известны.

*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 также 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 – распределенная система.

*Определение 3.1.4*

Грид-вычисления ([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) *grid* – решётка, сеть) — это форма [распределённых вычислений](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F). Здесь «виртуальный [суперкомпьютер](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D0%B5%D1%80%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80)» представлен в виде [кластеров](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80_(%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BE%D0%B2)), соединенных с помощью сети, слабосвязанных [гетерогенных](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D1%82%D0%B5%D1%80%D0%BE%D0%B3%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) компьютеров, работающих вместе для выполнения огромного количества заданий (операций, работ). Эта технология применяется для решения научных, математических задач, требующих больших вычислительных ресурсов.

*Определение 3.1.5*

Облачные вычисления ([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) *cloud computing*) — модель обеспечения удобного сетевого доступа по требованию к некоторому общему фонду конфигурируемых вычислительных ресурсов. Они могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами или обращениями к провайдеру.

Аппаратные и программные средства, реализующие распределенную модель вычислений.

*Определение 3.1.6.*

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 3.1.7*

Аппаратным средством модели распределенных вычислений могут служить грид-вычисления.

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 3.1.8*

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.

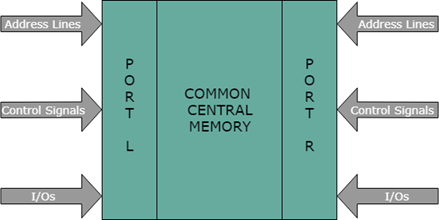
****

Рис. 3.1.1. Структура двухпортового статического ОЗУ

*Определение 3.1.9*

Зеркалирование 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 3.1.10*

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 3.1.11*

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).

## 3.2.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.

Существует два способа сделать программный код понятным для компьютера:

* компиляция;
* интерпретация.

*Определение 3.2.1*

Компилятор переводит в машинный код сразу всю программу и помещает его в память компьютера, не выполняя.

*Определение 3.2.2*

Интерпретатор переводит небольшой фрагмент исходной программы в машинный код. И только после того, как процессор выполнит этот код, он переходит к обработке следующего фрагмента.

*Определение 3.2.3*

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

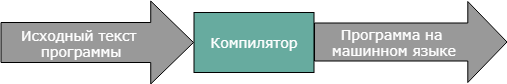
******

Рис.3.2.1. Компиляция исходных текстов

*Определение 3.2.4*

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

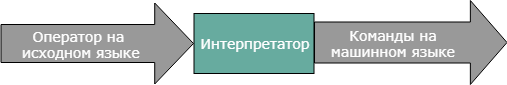
******

Рис.3.2.2. Интерпретация исходного текста программы

Основные различия компиляции и интерпретации:

* Компилятор работает со всей программой целиком, после чего выполняет ее. Интерпретатор делает это построчно.
* Интерпретатор прекратит свою работу после первой найденной ошибки.
* У компилятора анализ и обработка языка высокого уровня занимает больше времени чем у интерпретатора.
* В тоже время, компилятор выполняет обработанный код быстрее, нежели интерпретатор.

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

*Определение 3.2.5*

Низкоуровневое представление компьютерной программы в форме, включающей в себя информацию о процессорных командах, информацию о размещении программы, называется объектным кодом.

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

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

*Определение 3.2.6*

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 3.2.7*

Статическое связывание — метод, где части библиотеки копируются в исполняемый файл (см. *Определение 3.4.5*). Этот метод в сравнении с динамическим связыванием быстрее для запуска, а в некоторых случаях может быть более удобным для распространения и может сэкономить место.

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.

Байт-код используется современными языками программирования (особенно интерпретируемыми) для ускорения и облегчения работы интерпретатора. Метод трансляции в байт-код является промежуточным по эффективности между прямой интерпретацией и компиляцией в машинный код.

Длина любого кода операции равна одному байту. Например, инструкция — как раз однобайтовый код или байт-код. За ней уже могут следовать различные параметры, такие как номер регистра или адрес в памяти.

Программа байт-кода может быть выполнена путем анализа и непосредственного выполнения инструкций, по одной за раз. Преимуществами байт-кода являются большая эффективность и портируемость. Это означает, что определенный байт-код может исполняться на различных платформах и архитектурах, для которых реализован интерпретатор. Обычно эффективность интерпретации байт-кода выше, чем чистая интерпретация исходного кода или интерпретация АСД. Из-за простоты интерпретатора байт-кода его легко переносить на другую архитектуру компьютера.

Прямая оптимизация байт-кода: некоторые инструкции байт-кода намного трудозатратные, 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. Чтобы 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.3.2.3. Принцип работы байт-кода

## 3.3. Специализированные средства разработки программного обеспечения

*Определение 3.3.1*

Ассемблеры — компьютерные программы, которые переводят программы, написанные на языке ассемблера в машинные команды.

*Определение 3.3.2*

Трансляторы — технические средства, которые преобразуют программу в бинарный файл программы, состоящий из машинных команд, или выполняют непосредственно действия программы.

*Определение 3.3.3*

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 3.3.4*

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

*Определение 3.3.5*

Текстовые редакторы — компьютерные программы для создания, изменения, просмотра, вывода, поиска фрагментов текста.

*Определение 3.3.6*

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

*Определение 3.3.7*

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 3.3.8*

Профилирование — процесс анализа программы, нацеленный на ее оптимизацию. В ходе него обычно выясняют следующие характеристики:

* временную сложность программы;
* частоту вызовов функции;
* число верно спрогнозированных ветвлений;
* число кэш-промахов.

и др.

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

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

*Определение 3.3.9*

Точка останова (англ. *breakpoint*) — выбранная программистом позиция в коде, по достижении которой работа программы временно останавливается в целях отладки и тестирования. Потом она либо продолжает выполняться, либо завершается.

Точка останова может быть программной или поддерживаться процессором на аппаратном уровне. Всреде *Visual C++*, например, предусмотрена соответствующая инструкция для процессора. Для *Intel* это инструкция *int 3*, и ее выполнение приводит к исключению *EXCEPTION\_BREAKPOINT*.

Если разместить несколько точек останова, программа будет останавливаться через положенные промежутки времени. Так можно поэтапно проанализировать ее состояние: регистры, логи, файлы и т.д. Эту информацию выдает отладчик, считывая необходимые значения. Кроме того, именно отладчик оставляет по нужному адресу инструкцию, описанную выше. Хотя ее можно написать и вручную: для версий со встроенным ассемблером подойдет *\_\_asm int 3*; для других — функция *\_\_debugbreak()*.

При кодировании команды на ассемблере сначала пишется метка (если требуется), потом сама команда, а в конце операнды и комментарий. Команды содержатся в памяти по определенным адресам, а метки на эти адреса указывают. В современном ассемблере они мало чем отличаются от меток в С. В их названии (до 31 знака длиной) допустимы буквы, цифры и спецсимволы. При этом начинаться с цифр они не могут, символы «*$»* и «*?»* не рекомендуются, а регистр значения не имеет. Метки используются, например, 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 |

Примеры самих меток:COUNTER, PAGE69, $GR8.

Метке лучше дать имя, имеющее смысл. Оно не должно совпадать с зарезервированными именами, например, регистров AX и BX и т.д.

Метка, стоящая перед командой процессора, завершается двоеточием, *«:»*. Так ассемблер понимает, что нужно создать переменную с именем метки и значением адреса этой команды:

some\_loop:

lodsw; *считать слово из строки*

cmp ax,7; *если это 7* - *выйти из цикла*

loopne some\_loop

Метка, стоящая перед директивой ассемблера, уже является операндом этой директивы. Двоеточие не нужно:

codesg segment

lodsw; *считать слово из строки*

cmp ax,7; *если это 7 — выйти из цикла*

codesg ends

Некоторые метки уже предопределены, например, *«$»* возвращает текущий адрес. Посредством этой метки легко написать вечный цикл:

jmp $; *выполнять переход на себя*

Далее приведены директивы, работающие с метками.

1) Директива LABEL

С помощью этой директивы можно определить имя метки и ее тип. А типы бывают следующими:

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

В случае NEAR и FAR значению метки присваивается адрес следующей команды, в остальных случаях — значение следующих данных. Также метка получает тип, указанный явно.

2) Директива EQU

С помощью этой директивы можно определить значение, которое она связывает с именем метки. Это значение представляет собой адрес, целое число или строку:

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]

По своему назначению *EQU* похожа на директиву *#define* языка *C*. Она также отвечает за создание общих для программы параметров.

3) Директива =

С помощью этой директивы тоже определяют значение метки, но оно бывает только целочисленным. При этом такую метку можно переопределять.

## 3.4. Загрузчики и компоновщики. Форматы объектных модулей. Оверлейные загрузчики и оверлеи.

*Определение 3.4.1*

Загрузчик — составляющая операционной системы, которая позволяет загружать программы и библиотеки. Именно он помещает программы в память и готовит их к исполнению, поэтому в процессе запуска этап загрузки считается одним из важнейших. Этот этап включает в себя считывание данных из запускаемого файла; загрузку отсутствующих динамических библиотек и связывание адресов (в случае необходимости); создание нового процесса в памяти.

Количество обязанностей загрузчика может отличаться в зависимости от операционной системы. Например, некоторым из них требуется загрузчик, корректирующий адреса в исполняемом файле (см. *Определение 3.4.5*), потому что программа не всегда загружается в одно и то же место в адресном пространстве. Пусть существует программа, содержащая некоторые абсолютные адреса, которые имеют смысл, только когда она находится по определенному адресу A. В случае если эта программа загружена по другому адресу — B, нужно обновить все адреса, переведя их с B на A. Такое действие и называется связыванием адресов.

Загрузчик также может быть самостоятельной программой, которая позволяет запускать программы, скомпилированные для других ОС. Есть загрузчик и самой операционной системы.

*Определение 3.4.2*

Компоновщик (редактор связей, линкер) — утилита, которая принимает на вход объектные модули (см. *Определение 3.4.4*) и объединяет их в исполняемый файл, то есть совершает компоновку. Это требуется, если части компьютерной программы содержатся не в одном модуле, а ссылаются друг на друга с помощью таблицы символов.

*Определение 3.4.3*

Таблица символов — структура данных, которую заполняет компилятор каждым встреченным символом при компиляции кода. В ней хранится информация о переменной или функции: об их типе данных, области видимости и т.п. Эта таблица позволяет соотнести элементы объектного модуля с их адресами в памяти.

Символы могут быть определенными, неопределенными или локальными. Под определенными (экспортируемыми) чаще всего понимают имена функций и переменных, которые определены в данном модуле, и при этом ими могут пользоваться другие модули. Под неопределенными (импортируемыми) — функции и переменные, ссылающиеся на те модули, где они определены. И, наконец, локальные используются внутри объектного модуля, чтобы упростить процесс связывания адресов.

Компоновка бывает статической и динамической. При статической компоновщик записывает в файл программы все необходимые части. То есть код, требуемый для работы программы, целиком находится в ОЗУ. Например, пусть в системе одновременно работает N приложений, и все они пользуются несколькими одинаковыми функциями. Тогда в памяти будет храниться N копий таких функций. А при динамической компоновке эти функции находятся в памяти в единственном экземпляре; и параллельно работающие программы вызывают их из библиотек, а не из файлов программ. В библиотеках компоновщик может найти дефиниции тех символов, которые по умолчанию не определены, но на них ссылаются пользовательские объектные модули. Найденный модуль, в котором определен искомый символ, компоновщик либо включает в исполняемый файл (в случае статической компоновки), либо откладывает его до запуска программы (в случае динамической).

*Определение 3.4.4*

Объектный модуль (объектный файл) — это фрагмент машинного кода, который может использоваться программой или другими объектными модулями. Он является результатом работы компилятора и служит основой для получения исполняемого файла при помощи компоновщика.

Помимо собственных данных и перечня процедур, объектные модули хранят неопределенные адреса ссылок на подобную информацию в других модулях. Эти адреса вычисляет компоновщик. А именно: при вызове функции компилятор не помещает окончательный адрес программы в объектный файл; но он дает наставление компоновщику искать ссылку в таблицах всех обрабатываемых объектных модулей и уже там указывать окончательное местоположение.

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 3.4.5*

Исполняемый модуль (исполняемый файл) — файл, содержащий закодированную последовательность инструкций, которые способен выполнить компьютер. Исполняемые файлы бывают разных форматов, таких как ELF (используется в версиях unix), PE (в 32- и 64-разрядных Windows) и Mach-O (в случае macOS).

*Определение 3.4.6*

Overlay — метод программирования, при котором программы могут занимать больше места в памяти, чем предусмотрено системой. Также оверлеями называют фрагменты кода, на которые делят программу при реализации этого метода.

Процессору отводится ограниченное количество памяти. Оверлеи были придуманы для того, чтобы обеспечить эффективность загрузки ЦП. Идея заключается в том, что процесс можно поделить: пусть выполняется процесс A-E; и в данный момент времени требуется обработать только сегменты A, B, C; а после того, как с ними закончили, — D и E. Оверлеи помогают менять их местами между ЦП и памятью без дополнительной помощи со стороны операционной системы. Место в памяти, куда загружается оверлей, называется регионом*.* При загрузке в регион сохраняются все необходимые связи, например, с библиотеками.

# Глава 4. Параллельные вычисления

*Определение 4.1.1*

Параллельные вычисления — подход к организации компьютерных вычислений, в котором вычисления производятся параллельно и на разных конвейерах/ядрах/процессорах, но физически находящихся на одном компьютере.

При таком типе вычислений все узлы тесно соединены между собой, поэтому параметры их работы известны программисту.

Параллельные вычисления используют более одного вычислительного ядра, т.к. все управляющие потоки работают одновременно и занимают весь рабочий цикл ядра на время исполнения — параллельное вычисление невозможно на одноядерном компьютере.  
 Поскольку целью параллельных вычислений является ускорение выполнения алгоритма благодаря совмещению выполнения множества вычислительных операций — главная проблема при проектировании параллельных программ — обеспечить правильную последовательность взаимодействий между различными вычислительными процессами, а также координацию ресурсов, разделяемых между процессами. Это осуществляется с помощью применения обменных средств распараллеливания или же дополнительного выделения блоков алгоритма, предназначенных для выполнения параллельных вычислений, при этом обращаемых к разным областям общей памяти.

Параллельные машины различают по типу памяти — общая *(shared)* для всех процессоров и распределенная *(distributed)*, а также по типу управления — SIMD и MIMD (см. *Определение 1.3.2. и Определение 1.3.3.*).

Для SIMD архитектуры характерно наличие одного устройства управления, которое посылает инструкции всем остальным процессорам, причем у каждого процессора свой набор данных для работы. SIMD–системы, как правило, применяются для конкретных задач, которые требуют обычно не столько гибкости и универсальности вычислительной машины, сколько самой вычислительной силы.

В MIMD архитектуре к подобному классу систем относится большинство параллельных многопроцессорных вычислительных систем. У каждого процессора – собственный управляющий узел, а процессор может выполнять разные инструкции. Класс MIMD–машин имеет более широкий функционал, поэтому в пользовательских компьютерах применяются именно они.

Говоря об этих двух типах управления, в первую очередь стоит понимать, что они напрямую связаны с набором команд SSE и пришедшим ему на смену AVX, в результате чего SSE стал частным случаем AVX. Об этом мы уже упоминали ранее в главе 1.3. Сейчас же обратимся к современным версиям этих технологий — AVX2 и AVX-512.

*Определение 4.1.2*

AVX2 (*Advanced Vector Extensions 2*) и AVX-512 — развитие технологии AVX, 264-битные и 512-битные расширения для SIMD инструкций.

Поясним, спецификация AVX и AVX2 расширяет 128-битные регистры SSE XMM0…XMM15 до 256-битных регистров YMM0…YMM15.

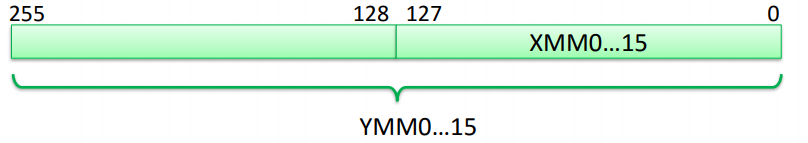
****

Рис.4.1. Регистры AVX

Каждый из 16 YMM регистров вмещает:

* 8 32-битных чисел с плавающей точкой или
* 4 64-битных числа двойной точности с плавающей точкой

А в AVX-512 же регистры SSE расширены до 512 разрядов, как показано на рисунке ниже, и их количество увеличено до 32 регистров ZMM0-ZMM31.

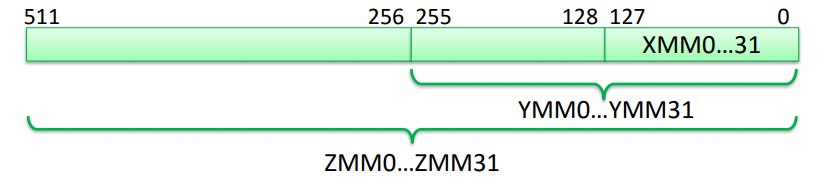
****

Рис.4.2. Регистры AVX2 и AVX-512.

Этот новый набор инструкций повышает производительность различных рабочих нагрузок, включая научное моделирование, финансовую аналитику, искусственный интеллект и глубинное обучение, 3D-моделирование и анализ данных, обработку изображений, аудио и видео, сжатие данных и шифрование. Впервые использован в 2013 г. в процессорах Intel на ядре Haswell.

Практическая польза для клиентского применения — прирост производительности при работе c мультимедийными данными, при этом нагрузка на процессор, естественно, снизится. Это касается и программ, в которых применяются алгоритмы распознавания голоса, лиц, жестов (только при использовании аппаратных ресурсов, совместимых с AVX2).

Далее перечислим новые возможности в AVX2 в сравнении с предыдущими технологиями:

* добавлена инструкция Fused Multiply Add (FMA), объединяющая в себе операции сложения и умножения (а это ключевые действия в цифровой обработке сигналов);
* реализована возможность операций с 256-битными целочисленными векторами;
* реализована поддержка так называемых gather-инструкций, благодаря которым перестает действовать строгое требование непрерывного расположения данных в памяти. Теперь данные могут собираться из разных адресов памяти;
* добавлены инструкции манипуляций/операций над битами.

В целом, инструкции AVX2 ориентированы на повышение быстродействия процессора в целочисленных вычислениях (при условии использования соответствующего программного обеспечения).

На смену AVX2 пришел новый набор инструкций AVX-512, в котором появилась концепция масочных регистров.

Opmask регистры

Большинство инструкций AVX-512 могут указывать на один из 8 opmask регистров (k0-k7). Первый (k0) является жестко запрограммированной константой, позволяющей исключать из операции отдельные элементы вектора. В большинстве инструкций opmask используется для управления, которые записывают значения адресату. Флаг управляет поведением opmask, который может быть либо «ноль», который зануляет все, что не выбрано маской, или «merge», что оставляет все, что было не тронуто. Поведение слияния идентично инструкциям смешивания. Регистры opmask обычно имеют ширину 16 бит, но могут быть до 64 бит с расширением AVX-512BW. Однако многие из битов фактически используются в зависимости от типа вектора маскированных команд. Для 32-битных однополярных или двойных слов 16 бит используются для маскировки 16 элементов в 512-битном регистре. Для double float и четырехзначных слов используются не более 8 бит маски.

AVX-512 позволяет более удобно работать с масочными регистрами и манипулировать с битами.

Сжатие и расширение

Инструкции сжатия и развертывания соответствуют операциям APL с тем же именем. Они используют маску немного иначе, чем другие инструкции AVX-512. Сжатие сохраняет только значения, отмеченные в маске, но сохраняет их сжатыми, пропуская и не резервируя место для немаркированных значений. Expand работает противоположным образом, загружая столько значений, сколько указано в маске, а затем распространяя их на выбранные позиции.

Инструкции с плавающей точкой

Введены три новые операции с плавающей точкой. Поскольку они не только новички в AVX-512, они имеют как упакованные / SIMD, так и скалярные версии.

Инструкции VFPCLASS проверяют, является ли значение с плавающей точкой одним из восьми специальных значений с плавающей точкой, какое из восьми значений будет запускать бит в регистре выходной маски, управляется непосредственным полем. Инструкции VRANGE выполняют минимальные или максимальные операции в зависимости от значения непосредственного поля, которое также может контролировать, выполняется ли операция абсолютно или нет, и отдельно, как обрабатывается знак. Инструкции VREDUCE работают с одним источником и вычитают из этого целую часть значения источника плюс количество битов, указанное в непосредственном поле его дроби.

# 

# Приложение

Специализированные алгоритмы и вычисления с повышенной точностью.

Существуют задачи, такие как 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. Метод Рунге-Ромберга, Ньютона, Лакс-Вендорфа, дихотомии – 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*

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

*Определение 2*

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.

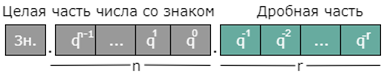
****

Рис. 1. 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.

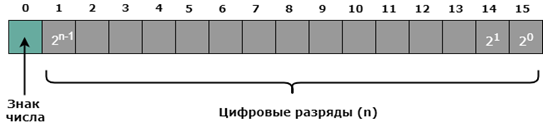
****

Рис. 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 – avx отрицательное). Знак их произведения: 0 + 1 = 1 ⇒ число отрицательное.

Ответ: ху = 1,1000111

# 

# Список сокращений

|  |  |  |
| --- | --- | --- |
| АВМ | — | аналоговые вычислительные машины |
| АЛУ | — | арифметико-логическое устройство |
| АЦП | — | аналого-цифровой преобразователь |
| ВМ | — | вычислительные машины |
| ВУ | — | внешним устройством |
| ГВМ | — | гибридные вычислительные машины |
| ИВВ | — | интерфейса ввода-вывода |
| ОВВ | — | окружение времени выполнения |
| ОЗУ | — | оперативное запоминающее устройство |
| ОСРВ | — | операционные системы реального времени |
| ПДП | — | прямой доступ к памяти |
| ПЗУ | — | постоянное запоминающее устройство |
| ПЭ | — | процессорный элемент |
| УВВ | — | устройства ввода-вывода |
| УУ | — | устройство управления |
| ФБ | — | функциональные блоки |
| ЦАП | — | цифро-аналоговый преобразователь |
| ЦВМ | — | цифровые вычислительные машины |
| ЦП | — | центральный процессор |
| ЦСП | — | цифровой сигнальный процессор |
| ЭВМ | — | электронная вычислительная машина |
| ЭВС | — | электронно-вычислительные средства |
| ADC | — | analog-to-digital converter (аналого-цифровой преобразователь) |
| AF | — | auxiliary carry flag (дополнительный флаг переноса) |
| AMD | — | advanced micro devices (американская компания) |
| ASIP | — | application-specific instruction-set processor (проблемно-ориентированный процессор) |
| AVX | — | advanced vector extensions (расширение системы команд процессора) |
| BIOS | — | basic input output system (базовая система ввода-вывода) |
| CF | — | carry flag (флаг переноса) |
| CISC | — | complex instruction set computing (компьютер с полным набором команд) |
| CNN | — | cable news network (кабельная новостная сеть) |
| CPU | — | central processing unit (центральный процессор) |
| DF | — | direction flag (флаг направления) |
| DRAM | — | dynamic random access memory (память динамического типа) |
| EDF | — | earliest deadline first (алгоритм планирования по ближайшему сроку завершения ) |
| ENOB | — | effective number of bits (эффективная разрядность) |
| EPIC | — | explicitly parallel instruction computing (вычисление с явным параллелизмом машинных команд) |
| FIFO | — | first-in, first-out («первым пришел, первым вышел») |
| FPU | — | floating point unit (сопроцессор с плавающей точкой) |
| FSB | — | front side bus (системная шина) |
| GDT | — | global descriptor table (глобальная таблица дескрипторов) |
| GPU | — | graphics processing unit (графический процессор) |
| HDD | — | hard disk drive (жесткий диск) |
| HTT | — | hyper-threading technology (гиперпоточная технология) |
| IBM | — | international business machines (название американской компании) |
| IF | — | interrupt flag (флаг прерывания) |
| ILP | — | instruction-level parallelism (параллелизм на уровне команд) |
| IOPL | — | input/output privilege level (флаг уровня приоритета ввода-вывода) |
| IPS | — | inner product step (шаг в скалярном произведении) |
| ISA | — | industry standard architecture (архитектура промышленного стандарта) |
| LSTF | — | least slack time first («чем меньше слабое время, тем выше приоритет») |
| MESI | — | modified exclusive shared invalid (протокол) |
| MIMD | — | multiple instruction stream, multiple data stream (множественный поток команд) |
| MISD | — | multiple instruction stream, single data stream (множественный поток команд,одиночный поток данных) |
| MMX | — | multimedia extensions (мультимедийные расширения) |
| MSP | — | multi-streaming processor (многопотоковый процессор) |
| NT | — | nested task (флаг вложенности задач) |
| OC | — | операционная система |
| OF | — | overflow flag (флаг переполнения) |
| PCI | — | peripheral component interconnect (дословно: взаимосвязь периферийных компонентов) |
| PF | — | parity flag (флаг четности) |
| POST | — | power on self test (процесс самопроверки компьютера) |
| PSW | — | program status word (регистра слова состояния программы) |
| RAM | — | random access memory (оперативное запоминающее устройство) |
| RISC | — | restricted (reduced) instruction set computer (компьютер с сокращенным набором команд) |
| RMS | — | rate monotonic scheduling (алгоритм с фиксированным приоритетом) |
| ROM | — | read only memory (постоянное запоминающее устройство) |
| RTE | — | run-time environment (окружение времени выполнения) |
| RTOS | — | real-time operating system (операционные системы реального времени) |
| SATA | — | serial advanced technology attachment (последовательный интерфейс) |
| SDRAM | — | synchronous dynamic random access memory (синхронная динамическая память с произвольным доступом) |
| SF | — | sign flag (знаковый флаг) |
| SIMD | — | single instruction stream & multiple data stream (одиночный поток команд и множественный поток данных) |
| SISD | — | single instruction stream & single data stream (одиночный поток команд и одиночный поток данных) |
| SMT | — | simultaneous multithreading (одновременная многопоточность) |
| SRAM | — | static random access memory (память статического типа) |
| SSD | — | solid-state drive (твердотельный накопитель) |
| SSE | — | streaming simd extensions (потоковое simd-расширение процессора) |
| SSP | — | single-streaming processor (отдельный потоковый процессор) |
| TF | — | trap flag (флаг трассировки) |
| TR | — | task register (регистр задачи) |
| TSS | — | task state segment (сегмент состояния задачи) |
| USB | — | universal serial bus (универсальная последовательная шина) |
| VLIW | — | very long instruction word («очень длинная машинная команда») |
| ZF | — | zero flag (флаг нуля) |