Операционные системы

Лекция 2

Архитектура вычислительных машин

План лекции

- Компоненты архитектуры, их назначение и взаимодействие
- Шинная архитектура
- Задачи операционной системы по управлению и организации работы компьютера
- Система прерываний

Архитектура вычислительных машин

В прошлой лекции мы выяснили основные концепции такого понятия как операционная система. Одной из функций любой операционной системы являлась:

«Предоставление прикладным программистам (и прикладным программам, естественно) вполне понятного абстрактного набора ресурсов взамен неупорядоченного набора **аппаратного обеспечения**»

Что же из себя представляет это аппаратное обеспечение и каковы причины становления той архитектуры, которую мы имеем на сегодняшний день?

Архитектура вычислительных машин

Для начала стоит задуматься, а что же такое архитектура вычислительных машин (далее – ВС)? – Ну, тут всё довольно не сложно, **архитектура ВС** – это совокупность основных устройств электронно-вычислительной машины (далее – ЭВМ) и способов их взаимодействия, определяющая общее построение аппаратных и программных средств

А какие устройства составляют современную ЭВМ?

На этот вопрос нельзя ответить однозначно для всех случаев. Однако существует базовый набор устройств который является жизненно важными для работы любого из устройств

А какой он, давайте и узнаем!

Архитектура вычислительных машин

Давайте начнём с того, что вспомним что современный компьютер в типичной конфигурации включает в себя:

- Системный блок (блок питания, корпус, материнская плата, процессор, жесткий диск, оперативная память, видеоадаптер, звуковой адаптер, приводы дисков)
- Монитор
- Клавиатура
- Мышь
- Периферийные устройства

Архитектура вычислительных машин

Если же говорить о жизненно важном, то получается следующее:

- Системный блок (**блок питания**, корпус, **материнская плата**, **процессор**, **жесткий диск***, **оперативная память**, видеоадаптер, звуковой адаптер, приводы дисков)
- Монитор
- Клавиатура
- Мышь
- Периферийные устройства

*Жёсткий диск здесь весьма условен, речь об устройстве постоянного хранения данных в целом

Архитектура вычислительных машин

Перед тем как приступить к рассмотрению отдельных устройств, стоит отметить что без некоторых из устройств **использование человеком** ЭВМ становится затруднительным или даже невозможным, однако это не значит что такая ЭВМ не может **функционировать!**

Примеры таких ЭВМ:

- Промышленные контроллеры
- Микроконтроллеры
- Бортовые компьютеры
- SCADA-системы

Процессоры

Центральный процессорное устройство (далее – ЦПУ) – это «мозг» компьютера. Он выбирает команды из памяти и выполняет их

Обычный цикл работы центрального процессора выглядит так: **выборка из памяти первой команды**, ее **декодирование** для определения ее типа и операндов, **выполнение** этой команды, а затем выборка, декодирование и выполнение последующих команд

Этот цикл повторяется до тех пор, пока не закончится программа. Таким образом программы выполняются

Для каждого типа центрального процессора существует определенный набор команд, которые он может выполнять

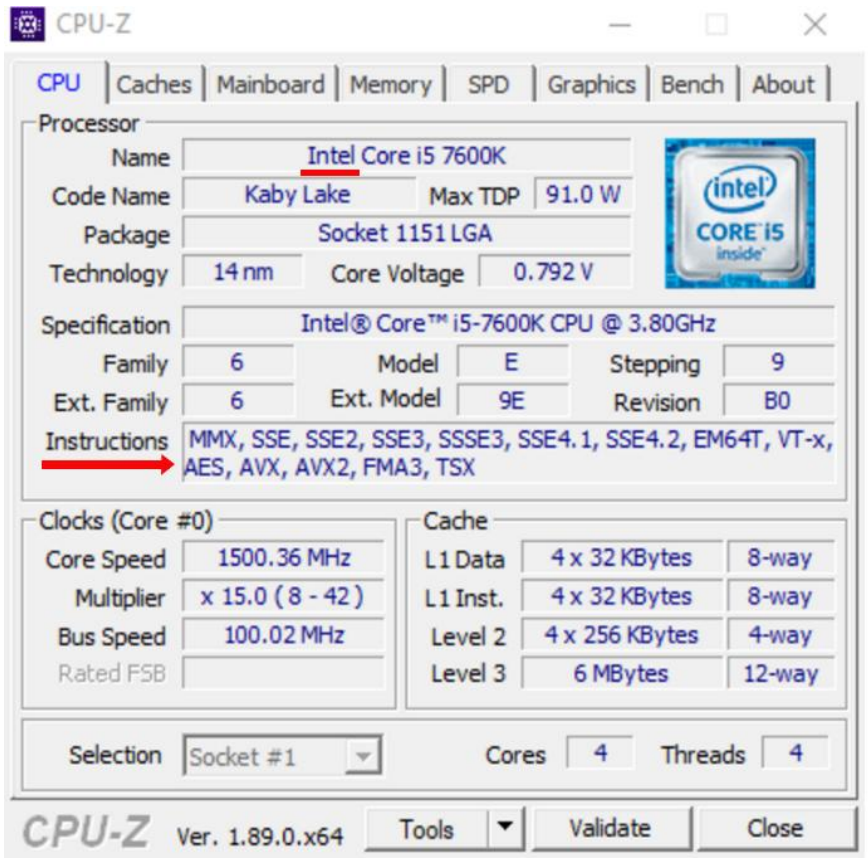
Процессоры

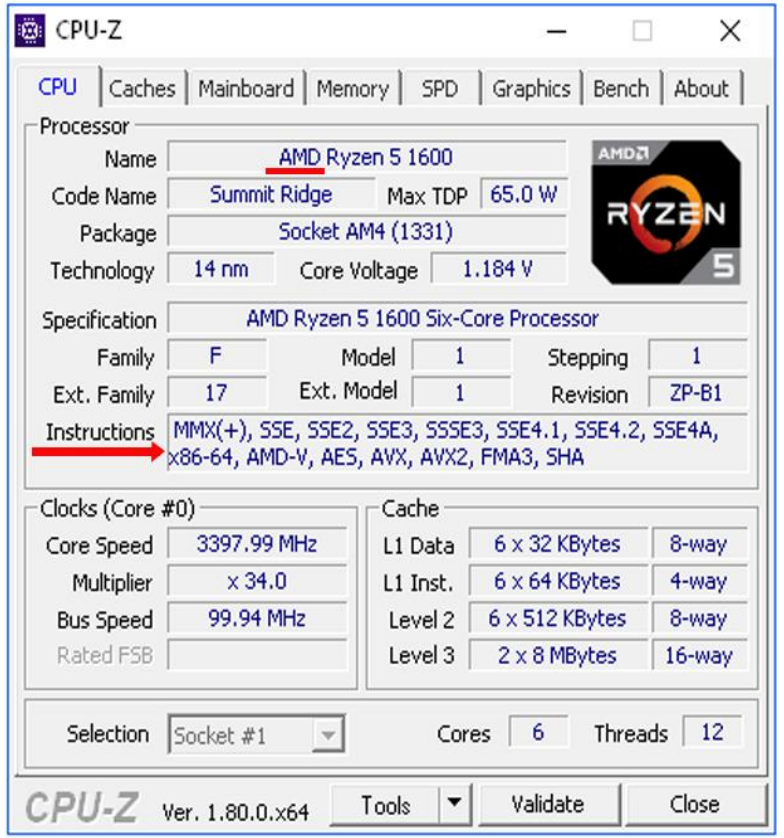
Вы можете посмотреть какие наборы команд поддерживает ваше ЦПУ с помощью соответствующих программ:

32-хразрядные: i386, x86, AMD

64-хразрядные: x64, AMD64

Поддержка двух разрядностей: x86-64





Процессоры

Поскольку доступ к памяти для получения команды или данных занимает намного больше времени, чем выполнение команды, у всех ЦПУ есть несколько собственных регистров для хранения основных переменных и промежуточных результатов

Соответственно набор команд содержит, как правило, команды на загрузку слова из памяти в регистр и на запоминание слова из регистра в память. Другие команды объединяют два операнда из регистров, памяти или обоих этих мест для получения результата – например, складывают два слова и сохраняют результат в регистре или в памяти

Процессоры

Среди специальных регистров ЦПУ стоит выделить:

- **Счетчик команд**
- **Указатель стека**
- **Слово состояния программы**

Что же это за регистры?

Счетчик команд (IP) – содержит адрес ячейки памяти со следующей выбираемой командой. После выборки этой команды счетчик команд обновляется, переставляя указатель на следующую команду

Процессоры

Указатель стека (SP) – ссылается на вершину текущего стека в памяти. Стек содержит по одному фрейму (области данных) для каждой процедуры, в которую уже вошла, но из которой еще не вышла программа. В стековом фрейме процедуры хранятся ее входные параметры, а также локальные и временные переменные, не содержащиеся в регистрах

Слово состояния программы (PSW) – содержатся биты кода условия, устанавливаемые инструкциями сравнения, а также биты управления приоритетом центрального процессора, режимом (пользовательским или ядра) и другие служебные биты

Регистр PSW играет важную роль в системных вызовах и операциях ввода-вывода

Процессоры

Операционная система должна все знать о состоянии всех регистров. При временном мультиплексировании (умный способ говорить **разделение времени**) ЦПУ операционная система может останавливать работающую программу, чтобы запустить или возобновить работу другой программы

При каждой остановке работающей программы операционная система должна сохранять состояние всех регистров, чтобы восстановить его при последующем возобновлении работы этой программы

Об этом всём мы ещё поговорим при обсуждении процессов!

Процессоры

Среди важных аспектов устройства и принципов работы ЦПУ которые повлияли на устройство операционных систем хотелось бы отметить:

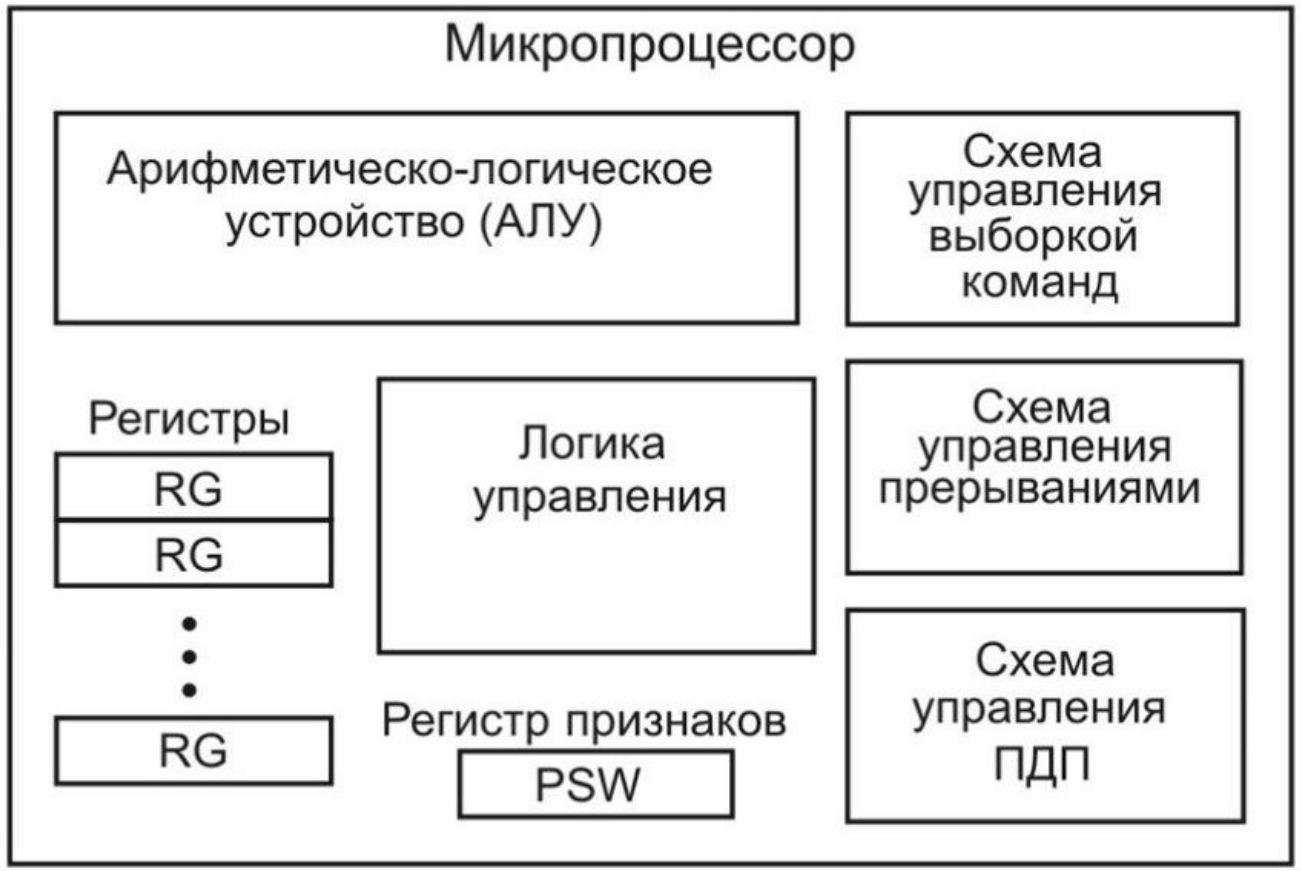
- **Вычислительный конвейер**
- **Суперскалярность**
- **Многоядерность**
- **Многопоточность (Гиперпоточность)**

Если мы говорим о разработке прикладных приложений то зачастую данные аспекты обходят разработчика стороной, однако если мы говорим о создании операционной системы или модулей для неё, то не учитывать данные особенности невозможно

Процессоры

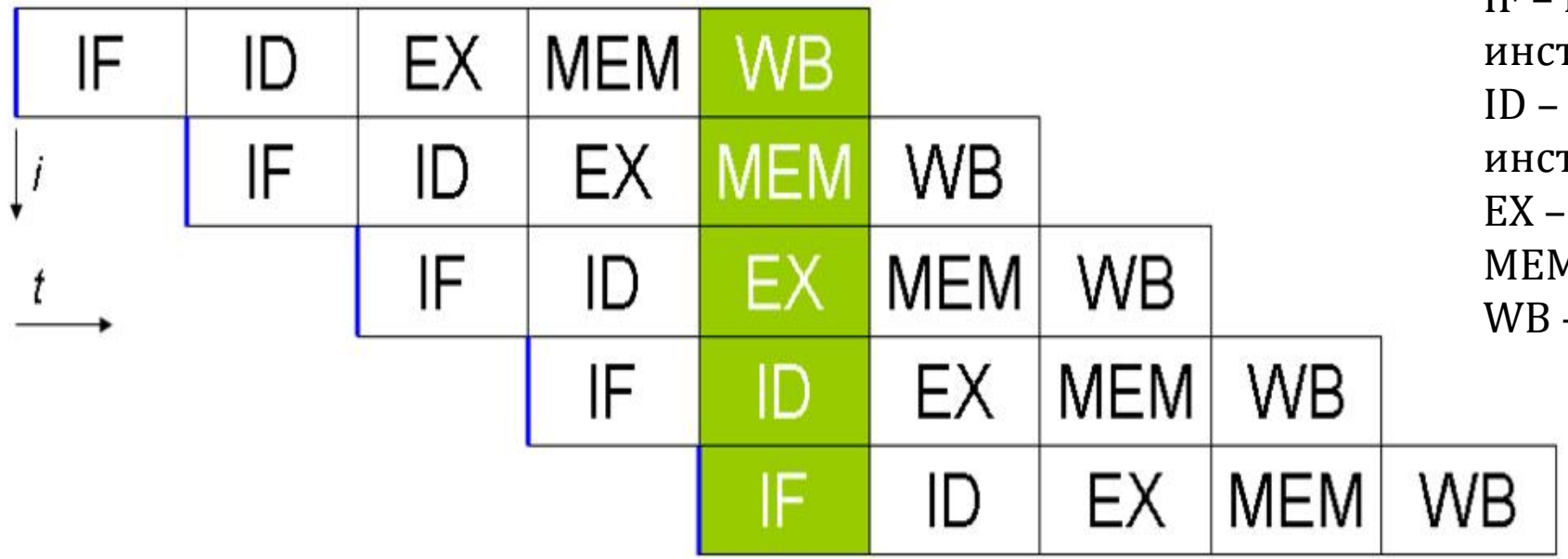
Перед кратким объяснением данных аспектов устройства ЦПУ, хотелось бы представить как он устроен в целом:

На данном рисунке представлен пример структуры одного **ядра центрального процессора –** самостоятельной вычислительной единицы процессора



Процессоры

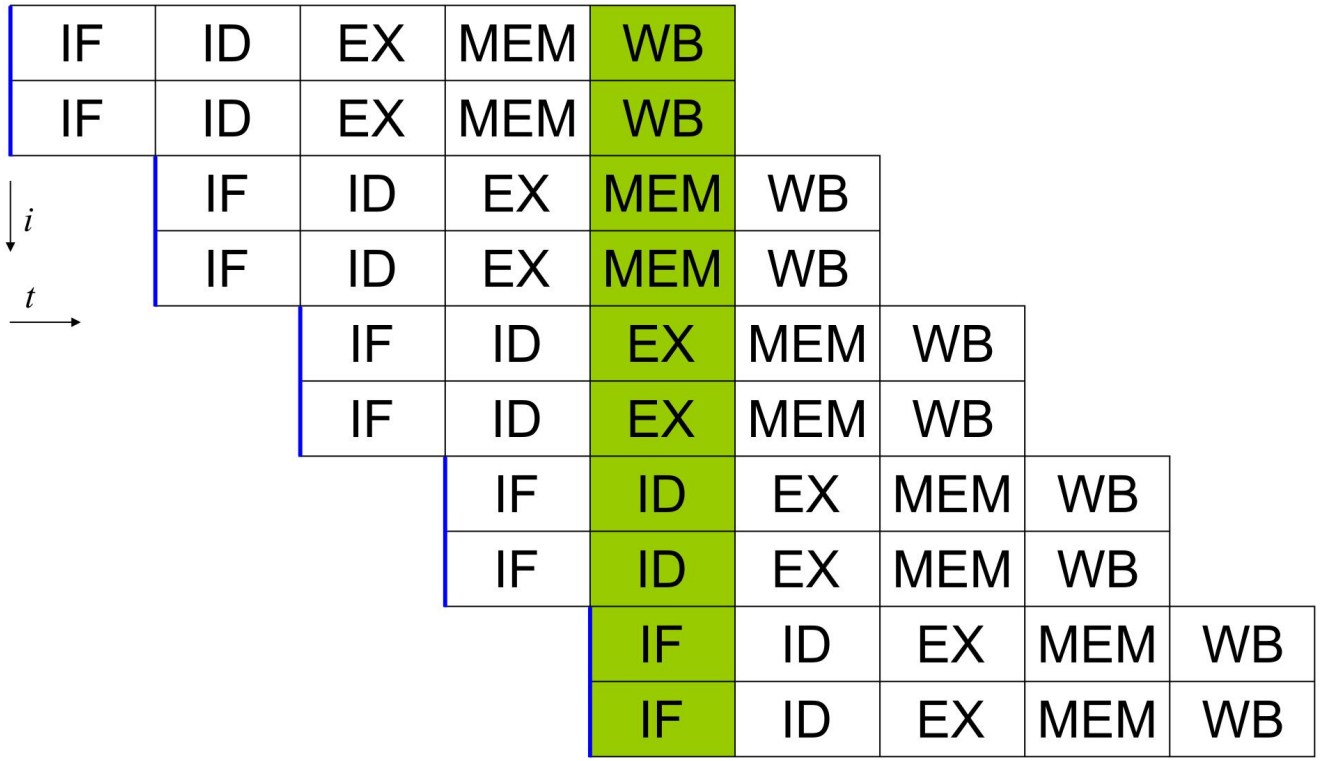
Вычислительный конвейер – это подход к организации работы процессора при котором за один такт выполняется более одной команды за счёт одновременной работы узлов отвечающих за разные этапы обработки команд



IF – получение инструкции
ID – раскодирование инструкции
EX – выполнение
MEM – доступ к памяти
WB – запись в регистр

Процессоры

Суперскалярность – это подход к организации работы процессора при котором за один такт выполняется более одной команды за счёт включения в процессор множества одинаковых узлов отвечающих за одни и те же этапы обработки команд (например, много АЛУ, дешифраторов и т.д.)



Процессоры

Многоядерность – это подход к организации работы ЦПУ при котором в его состав включается несколько физических ядер, которые могут выполнять команды различных потоков по настоящему параллельно

Многопоточность – это технология которая позволяет ядру процессора сохранять состояние двух различных потоков и переключаться между ними за наносекунды

Например, если одному из процессов нужно прочитать слово из памяти (что занимает несколько тактов), многопоточный процессор может переключиться на другой поток

Многопоточность не предлагает настоящей параллельной обработки данных. Одновременно работает только один процесс, но время переключения между процессами сведено до наносекунд

Процессоры

Как уже упоминалось, большинство центральных процессоров, за исключением самых простых, используемых во встраиваемых системах, имеют два режима работы: **режим ядра** и **пользовательский режим** (режим пользователя)

Обычно режимом управляет специальный бит в слове состояния программы – PSW. При работе в режиме ядра процессор может выполнять любые команды из своего набора и использовать любые возможности аппаратуры

На настольных и серверных машинах операционная система обычно работает в режиме ядра, что дает ей доступ ко всему оборудованию

На большинстве встроенных систем в режиме ядра работает только небольшая часть операционной системы, а вся остальная ее часть – в режиме пользователя

Процессоры

Пользовательские программы всегда работают в режиме пользователя, который допускает выполнение только подмножества команд и дает доступ к определенному подмножеству возможностей аппаратуры

Как правило, в пользовательском режиме запрещены все команды, касающиеся операций ввода-вывода и защиты памяти. Также, разумеется, запрещена установка режима ядра за счет изменения значения бита режима PSW

Для получения услуг от операционной системы пользовательская программа должна осуществить **системный вызов**, который перехватывается внутри ядра и вызывает операционную систему

Процессоры

Инструкция перехвата **TRAP** осуществляет переключение из пользовательского режима в режим ядра и запускает операционную систему. Когда обработка вызова будет завершена, управление возвращается пользовательской программе и выполняется команда, которая следует за системным вызовом

Вопрос этих двух режимов работы ЦПУ более подробно рассмотрим в следующей лекции вместе с ядром ОС

Механизм же лежащий в основе системных вызовов рассмотрим далее по лекции

Процессоры

Заканчивая тему процессоров можно рассмотреть какие характеристики являются ключевыми для ЦПУ, а именно:

- Количество ядер (и дополнительные характеристики)
- Максимальное количество потоков
- Сокет
- Базовая и максимальная тактовая частота
- Объем и уровни кэша
- Типы поддерживаемой памяти (DDR4, DDR5)
- Количество каналов памяти
- Версия PCI Express
- Встроенная графика
- Техпроцесс
- Типичное тепловыделение

Память

Второй основной составляющей любой ЭВМ является **память**

В идеал память должна быть максимально быстрой (работать быстрее, чем производится выполнение одной инструкции, чтобы работа центрального процессора не замедлялась обращениями к памяти), довольно большой и чрезвычайно дешевой

Никакая современная технология не в состоянии удовлетворить все эти требования, поэтому используется другой подход

Система памяти создается в виде **иерархии уровней**. Верхние уровни обладают более высоким быстродействием, меньшим объемом и более высокой удельной стоимостью хранения одного бита информации, чем нижние уровни, иногда в миллиарды и более раз

Память

Выглядит подобная иерархия подобным образом:

МПП – микропроцессорная память (регистры)

Память

Верхний уровень состоит из внутренних регистров процессора. Они выполнены по той же технологии, что и сам процессор, и поэтому не уступают ему в быстродействии. Следовательно, к ним **нет и задержек доступа** (один такт)

Затем следует кэш-память, которая управляется главным образом аппаратурой

Когда программе нужно считать слово из памяти, аппаратура кэша проверяет, нет ли нужной строки в кэш-памяти. Если строка в ней имеется, то происходит результативное обращение к кэш-памяти (cache hit – кэш-попадание), запрос удовлетворяется за счет кэш-памяти без отправки запроса по шине к оперативной памяти. Обычно результативное обращение к кэшу занимает по времени два такта

Память

Следующей в иерархии идёт **оперативная память**. Это главная рабочая область системы памяти машины. Оперативную память часто называют **оперативным запоминающим устройством** (ОЗУ), или памятью с произвольным доступом (Random Access Memory (RAM))

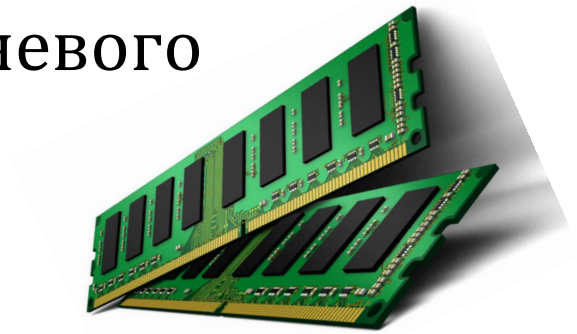
Все запросы процессора, которые не могут быть удовлетворены кэш-памятью, направляются к оперативной памяти

Дополнительно к оперативной памяти многие компьютеры оснащены небольшой по объему неизменяемой памятью с произвольным доступом – **постоянным запоминающим устройством** (ПЗУ), оно же память, предназначенная только для чтения (Read Only Memory (ROM))

Память

В отличие от ОЗУ она не утрачивает своего содержимого при отключении питания, то есть является энергонезависимой. ПЗУ программируется на предприятии-изготовителе и впоследствии не подлежит изменению. Эта разновидность памяти характеризуется высоким быстродействием и дешевизной

На некоторых компьютерах в ПЗУ размещается начальный загрузчик, используемый для их запуска. Такой же памятью, предназначенной для осуществления низкоуровневого управления устройством, оснащаются некоторые контроллеры ввода-вывода



Память

Еще одна разновидность памяти – **CMOS-память**, которая является энергозависимой

Во многих компьютерах CMOS-память используется для хранения текущих даты и времени. CMOS-память и схема электронных часов, отвечающая за отсчет времени, получают питание от миниатюрной батарейки (или аккумулятора), поэтому значение текущего времени исправно обновляется, даже если компьютер отсоединен от внешнего источника питания

Потребление энергии CMOS памятью настолько низкое, что батарейки, установленной на заводе-изготовителе, часто хватает на несколько лет работы

Однако когда батарейка начинает выходить из строя, на компьютере могут проявиться признаки «болезни Альцгеймера» и он станет «забывать» то, что помнил годами, например с какого жесткого диска следует производить загрузку

Память

Следующим после оперативной памяти уровнем нашей иерархии памяти является **магнитный (жесткий) диск**.

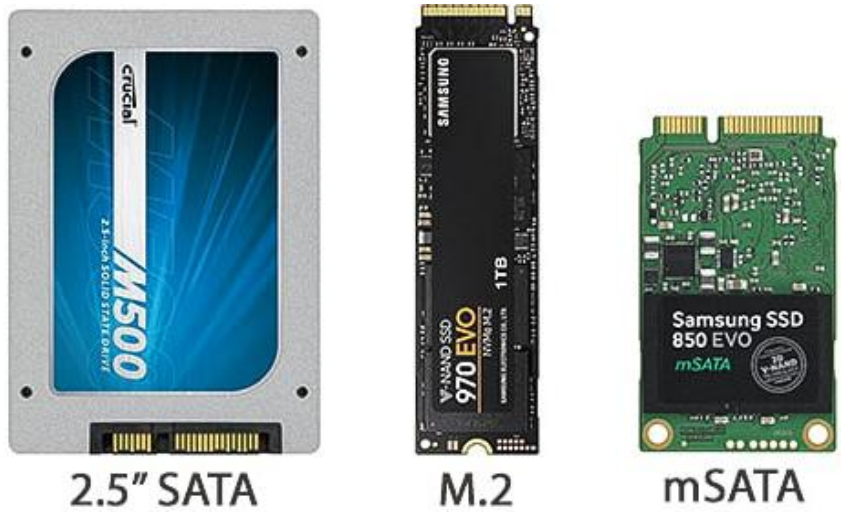
Дисковый накопитель в пересчете на бит информации на два порядка дешевле, чем ОЗУ, а его емкость зачастую на два порядка выше. Единственная проблема состоит в том, что время произвольного доступа к данным примерно на три порядка медленнее. Причина в том, что диск является механическим устройством

Об их устройстве поговорим позже, когда будем обсуждать работу с файловыми системами

Память

Порой речь заходит о таких дисках, которые на самом деле дисками не являются, например о **твердотельных накопителях – SSD** (Solid State Disks)

У них нет движущихся частей, дисковых пластин, а данные хранятся во флеш-памяти. Они напоминают диски только тем, что содержат большой объем данных, который при отключении питания не теряется



2.5" SATA M.2 mSATA

Память

Многие компьютеры поддерживают схему, которая называется **виртуальной памятью**. Ее мы довольно основательно рассмотрим в будущих лекциях. Она дает возможность запускать программы, превышающие по объему физическую память компьютера, за счет помещения их на диск и использования оперативной памяти как некой разновидности кэша для наиболее интенсивно исполняемых частей

Эта схема требует прозрачного для программы преобразования адресов памяти, чтобы конвертировать адрес, сгенерированный программой, в физический адрес, по которому слово размещено в ОЗУ

Такое отображение адресов осуществляется частью центрального процессора, называется **блоком управления памятью** (Memory Management Unit (MMU)), или диспетчером памяти

Память

Использование **кэширования** и **MMU** может оказать существенное влияние на производительность. При работе в мультипрограммном режиме, когда осуществляется переключение с одной программы на другую, иногда называемое **переключением контекста** (context switch), может потребоваться сброс всех измененных блоков из кэш-памяти и изменение регистров отображения в MMU

Обе эти операции обходятся слишком дорого, и программисты всеми силами стараются их избежать

Устройства ввода-вывода

Центральный процессор и память не единственные ресурсы, которыми должна управлять операционная система. С ней также активно взаимодействуют и **устройства ввода-вывода** информации

Устройства ввода-вывода обычно состоят из двух компонентов: **самого устройства** и **его контроллера**.

Контроллер представляет собой микросхему или набор микросхем, которые управляют устройством на физическом уровне. Он принимает от операционной системы команды, например считать данные с помощью устройства, а затем их выполняет

Устройства ввода-вывода

Довольно часто непосредственное управление устройством очень сложно и требует высокого уровня детализации, поэтому задачей контроллера является предоставление операционной системе простого (но не упрощенного) интерфейса

Например, контроллер диска может получить команду прочитать сектор 11 206 с диска 2. Получив команду, контроллер должен преобразовать этот простой порядковый номер сектора в номер цилиндра, сектора и головки

Операция преобразования может быть затруднена тем, что внешние цилиндры имеют больше секторов, чем внутренние, а номера «плохих» секторов отображаются на другие секторы

Устройства ввода-вывода

Затем контроллер должен определить, над каким цилиндром сейчас находится привод головок, и выдать команду, чтобы он переместился вперед или назад на требуемое количество цилиндров

Далее необходимо дождаться, пока нужный сектор не попадет под головку, а затем приступить к чтению и сохранению битов по мере их поступления из привода, удаляя заголовки и подсчитывая контрольную сумму. В завершение он должен собрать поступившие биты в слова и сохранить их в памяти

Для осуществления всей этой работы контроллеры часто содержат маленькие встроенные компьютеры, запрограммированные на выполнение подобных задач

Устройства ввода-вывода

Другим компонентом является само устройство. Устройства имеют довольно простые интерфейсы, поскольку они, во-первых, обладают весьма скромными возможностями, а во-вторых, должны отвечать общим стандартам

Соблюдение последнего условия необходимо для того, чтобы, к примеру, любой контроллер SATA-диска смог работать с любым SATA-диском

Поскольку интерфейс устройства скрыт его контроллером, все операционные системы видят только **интерфейс контроллера**, который может существенно отличаться от **интерфейса самого устройства**

Устройства ввода-вывода

Так как все типы контроллеров отличаются друг от друга, для управления ими требуется различное программное обеспечение.

Программа, предназначенная для общения с контроллером, выдачи ему команды и получения поступающих от него ответов, называется **драйвером устройства**

Каждый производитель контроллеров должен поставлять вместе с ними драйверы для каждой поддерживаемой операционной системы

Для использования драйвер нужно поместить в операционную систему, предоставив ему тем самым возможность работать в режиме ядра (не всегда)

Устройства ввода-вывода

Существует три способа установки драйвера в ядро:

- Компоновка ядра ОС вместе с драйвером
- Указание необходимых драйверов в конфигурационных файлах с последующей перезагрузкой для их подгрузки
- Динамическая загрузка драйверов (например, в технологии plug and play), т.е. непосредственно в момент работы ОС

Ввод и вывод данных можно делать также тремя различными способами:

- Через системные вызовы
- Через механизм прерываний
- Через прямой доступ к памяти

Устройства ввода-вывода

В простейшем из них пользовательская программа производит **системный вызов**, который транслируется ядром в процедуру вызова соответствующего драйвера

После этого драйвер приступает к процессу ввода-вывода. В это время он выполняет очень короткий цикл, постоянно опрашивая устройство и отслеживая завершение операции (обычно занятость устройства определяется состоянием специального бита)

По завершении операции ввода-вывода драйвер помещает данные (если таковые имеются) туда, куда требуется, и возвращает управление. Затем операционная система возвращает управление вызывающей программе

Этот способ называется **активным ожиданием** или **ожиданием готовности**, а его недостаток заключается в том, что он загружает процессор опросом устройства об окончании работы

Устройства ввода-вывода

Второй способ заключается в том, что драйвер запускает устройство и просит его выдать **прерывание** по окончании выполнения команды (завершении ввода или вывода данных)

Сразу после этого драйвер возвращает управление. Затем операционная система блокирует вызывающую программу, если это необходимо, и переходит к выполнению других задач

Когда контроллер обнаруживает окончание передачи данных, он генерирует **прерывание**, чтобы просигнализировать о завершении операции

Система прерываний

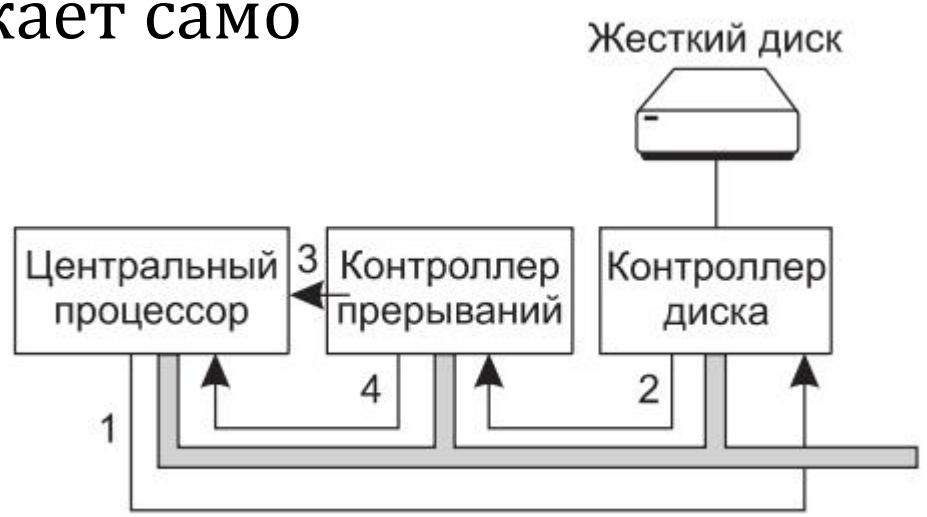
Прерывание – одна из базовых концепций вычислительной техники, которая заключается в том, что при наступлении какого-либо события происходит передача управления специальной процедуре, называемой **обработчиком прерываний** (ISR, Interrupt Service Routine)

В отличие от условных и безусловных переходов, прерывание может быть вызвано в любом месте программы, в том числе если выполнение программы приостановлено, и обусловлено обычно внешними по отношению к программе событиями. После выполнения необходимых действий, обработчик прерываний, как правило, возвращает управление прерванной программе

Система прерываний

Прерывания играют очень важную роль в работе операционной системы, поэтому рассмотрим их более подробно

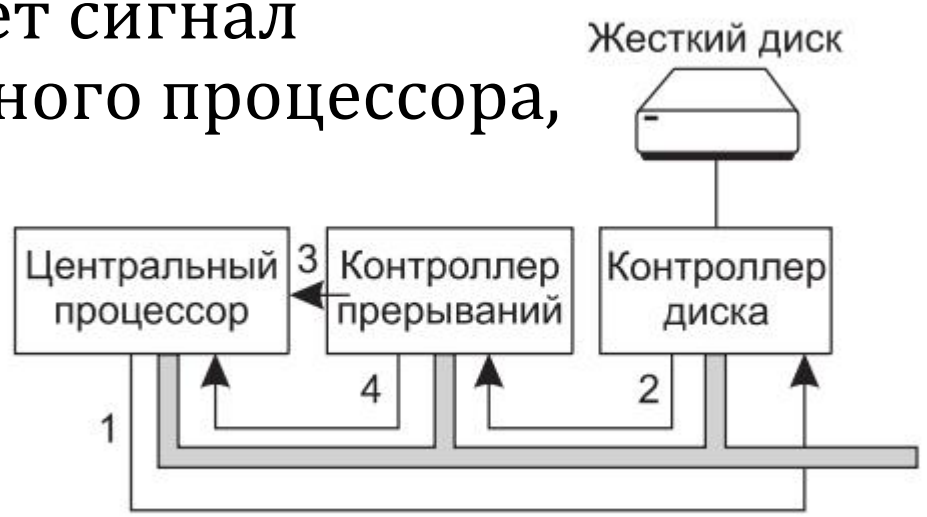
На первом этапе (1) драйвер передает команду контроллеру, записывая информацию в его регистры. Затем контроллер запускает само устройство



Система прерываний

На втором этапе (2), когда контроллер завершает чтение или запись заданного ему количества байтов, он выставляет сигнал для микросхемы контроллера прерываний, используя для этого определенные линии шины

На третьем этапе (3), если контроллер прерываний готов принять прерывание, он выставляет сигнал на контакте микросхемы центрального процессора, информируя его о завершении операции

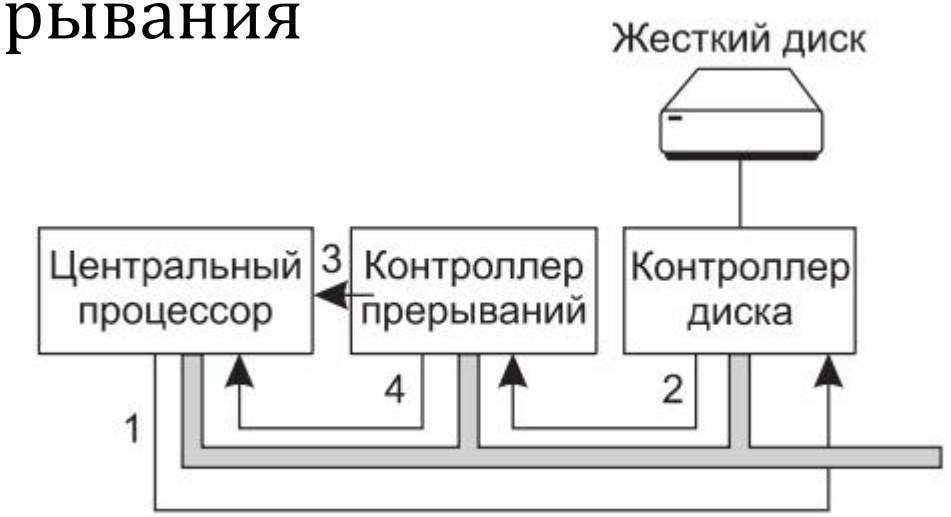


Система прерываний

На четвертом этапе (4) контроллер прерываний выставляет номер устройства на шину, чтобы процессор мог его считать и узнать, какое устройство только что завершило работу (поскольку одновременно могут работать сразу несколько устройств)

Дальше происходит обработка прерывания непосредственно процессором

Более наглядную версию данного алгоритма смотрите в конце лекции



Система прерываний

Если кратко, то обработка происходит так:

1. Сохранение информации о текущей выполняемой программе в стеке и переход в режим ядра

2. Поиск и выполнение обработчика прерываний

3. Возвращение управления ранее работавшей пользовательской программе

Поиск обработчика прерываний происходит в специальной области памяти называемой **вектором прерываний**

Система прерываний

Также стоит упомянуть какие бывают прерывания

В зависимости от источника, прерывания делятся на:

- **Аппаратные** – возникают как реакция микропроцессора на физический сигнал от некоторого устройства (клавиатура, системные часы, клавиатура, жесткий диск и т.д.), по времени возникновения эти прерывания асинхронны, т.е. происходят в случайные моменты времени
- **Программные** – вызываются искусственно с помощью соответствующей команды из программы (**int**), предназначены для выполнения некоторых действий операционной системы, являются синхронными
- **Исключения** – являются реакцией микропроцессора на нестандартную ситуацию, возникшую внутри микропроцессора во время выполнения некоторой команды программы (деление на ноль, прерывание по флагу TF (трассировка))

Система прерываний

Общая классификация прерываний:

- **Внешние** – вызываются внешними по отношению к микропроцессору событиями (это группа аппаратных прерываний)
- **Внутренние** – возникают внутри микропроцессора во время вычислительного процесса (по существу это исключительные ситуации и программные прерывания)

Внешние прерывания подразделяются на не-маскируемые и маскируемые (маскирование прерывания = игнорирование):

- **Маскируемые** прерывания генерируются контроллером прерываний по заявке определенных периферийных устройств
- **Немаскируемые** прерывания инициируют источники, требующие безотлагательного вмешательства со стороны микропроцессора

Система прерываний

Особые ситуации (по сути синхронные прерывания), генерируемые процессором, подразделяются на три типа – ошибки, ловушки и сбои. В зависимости от типа особой ситуации различается реакция процессора на ее возникновение

Ошибка (Fault) – это особая ситуация, которая может быть исправлена обработчиком особой ситуации. При встрече ошибки состояние процессора сохраняется в том виде, каким оно было до начала выполнения команды, инициировавшей генерацию ошибки, а значения CS:EIP, указывающие на эту команду сохраняются в стеке обработчика. Прерванная программа после исправления ошибки может быть продолжена непосредственно с команды, вызвавшей эту ошибку

Система прерываний

Ловушка (Trap) – особая ситуация, которая генерируется после выполнения соответствующей команды. В этом случае сохраняемые в стеке значения CS:EIP, указывают на команду, которая будет выполняться вслед за командой, вызвавшей ловушку; например, если ловушка произошла во время команды JMP, то сохраненные значения CS:EIP указывают на команду, являвшуюся целью команды JMP

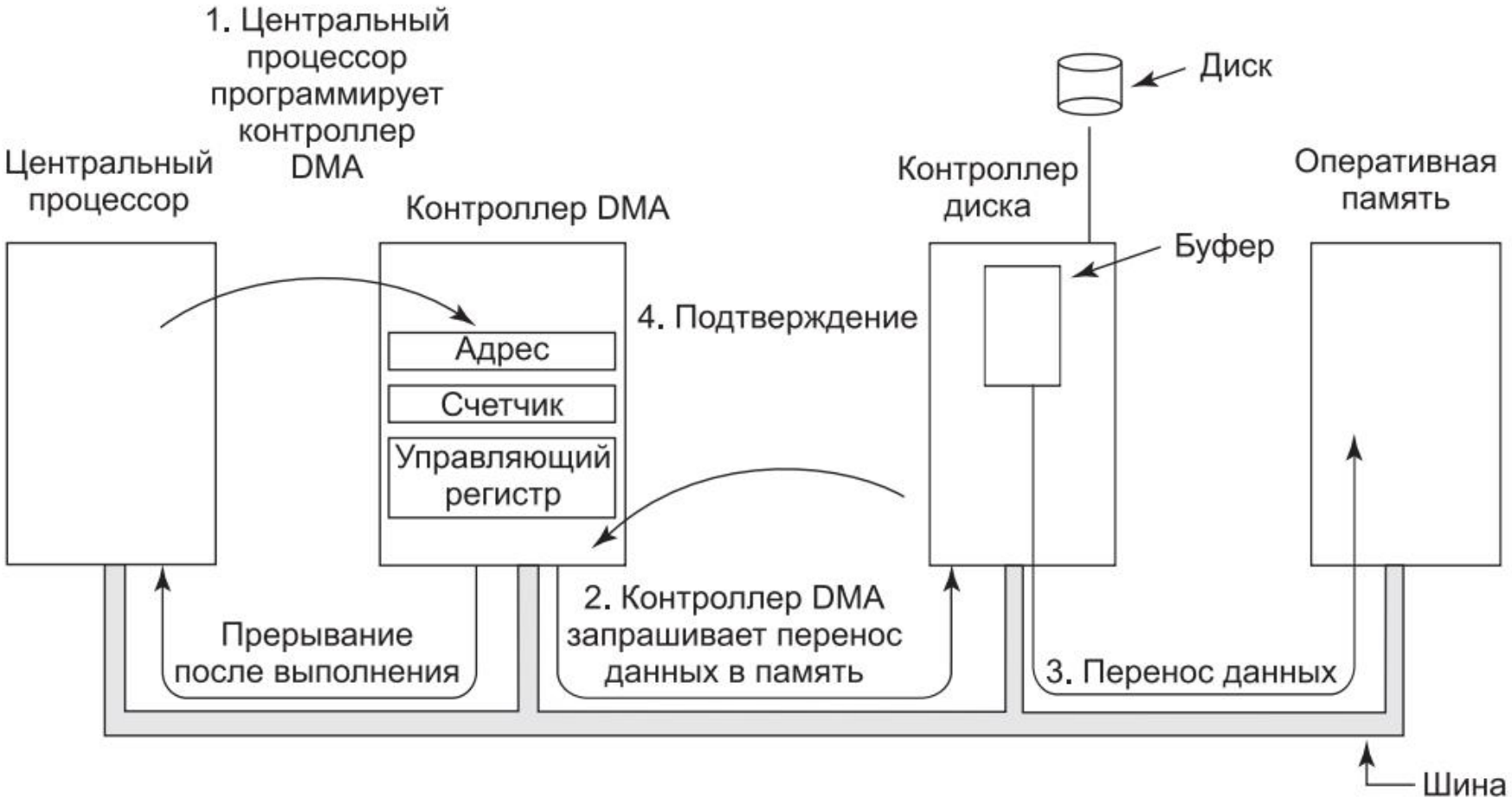
Сбой (Abort) – это особая ситуация, которая не допускает точную локализацию вызвавшей ее команды и не допускает перезапуска. Сбои используются для сообщений о некоторых ошибках, таких как: технические неисправности и наличие некорректных значений в системных таблицах

Устройства ввода-вывода

При третьем способе ввода-вывода используется специальный контроллер **прямого доступа к памяти** (Direct Memory Access (DMA)), который может управлять потоком битов между оперативной памятью и некоторыми контроллерами без постоянного вмешательства центрального процессора

Центральный процессор осуществляет настройку контроллера DMA, сообщая ему, сколько байтов следует передать, какое устройство и адреса памяти задействовать и в каком направлении передать данные, а затем дает ему возможность действовать самостоятельно. Когда контроллер DMA завершает работу, он выдает запрос на прерывание, который обрабатывается в ранее рассмотренном порядке

Устройства ввода-вывода



Источники питания

Рассмотрев три основных компонента любой ЭВМ, можно упомянуть и про источники питания, ведь без них никакая ЭВМ не запустится, будь у вас хоть все возможные остальные компоненты

Электронные компоненты компьютера требуют питания постоянного тока напряжением 3.3, 5 и 12 вольт, в то время к в электрической сети 220 вольт переменного тока

Необходимые преобразования выполняет **блок питания**

При выборе блока питания важнейший параметр – мощность Нужно сложить потребляемую мощность всех компонентов компьютера и оставить некоторый резерв

Материнская плата

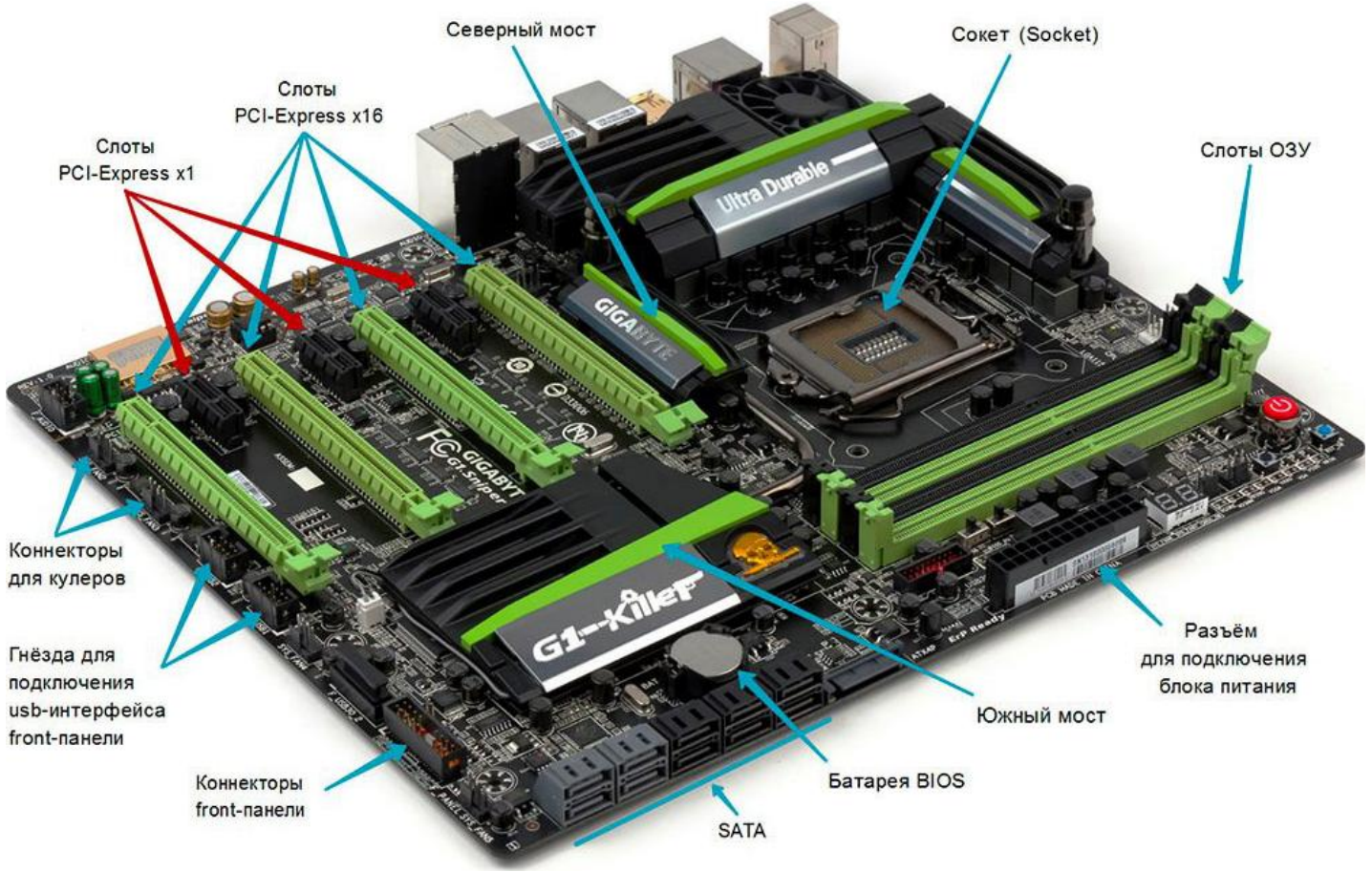
Но как же нам их объединить в полноценную ЭВМ? Ведь по отдельности все эти элементы ничего не стоят, тогда что?

Для этого нам на помощь приходит **материнская плата**

Материнская плата отвечает за организацию взаимодействия между установленными на ней устройствами, обеспечивая их нужным напряжением и необходимыми частотными характеристиками. На ней расположены слоты и разъемы для подключения других компонентов компьютера: видеокарты, оперативной памяти, процессора, накопителей данных и т. д.

Материнская плата

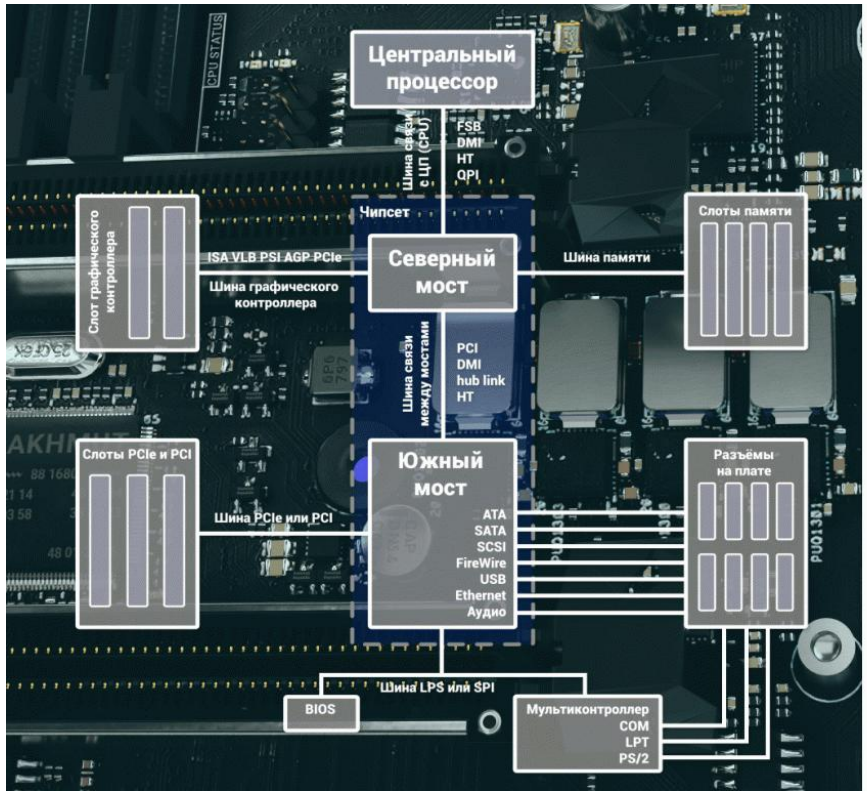
Вот такие они, эти ваши материнские платы:



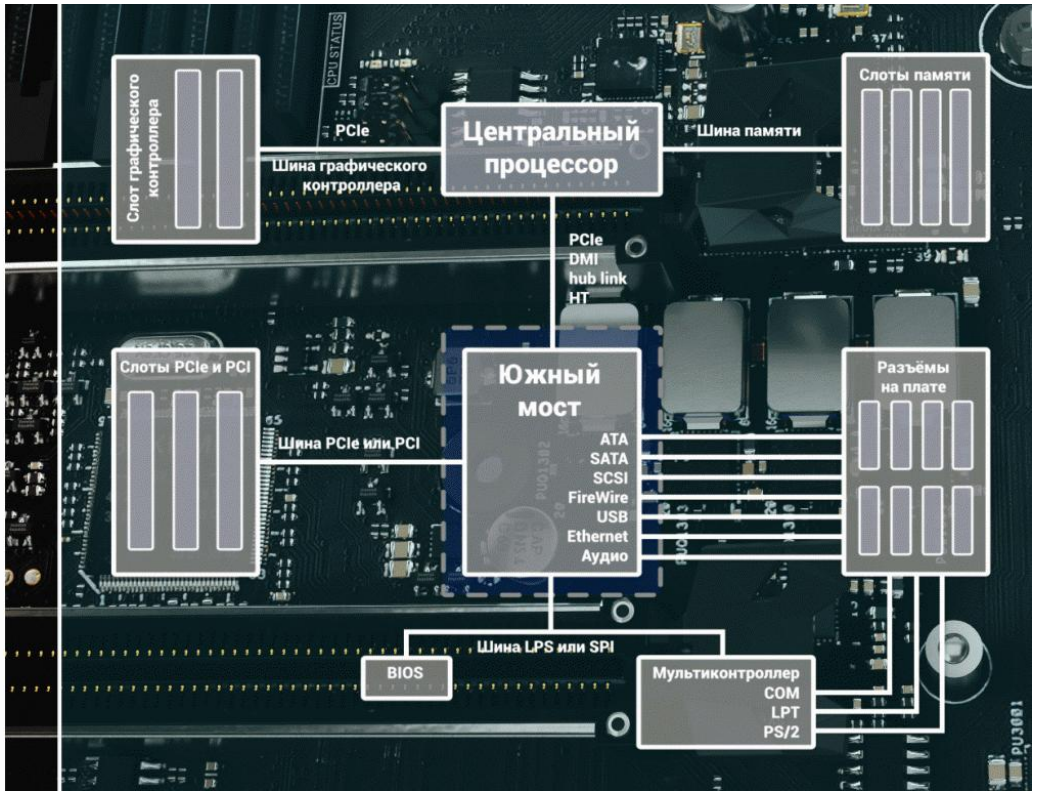
Материнская плата

Изобразив схематично устройство такой платы можем получить следующее:

Классическая



Современная



Архитектура вычислительных машин

Понимая что материнская плата предназначена для налаживания взаимодействия между компонентами, можно задаться вопросом, а как именно? И что же привело к тому, что мы имеем сегодня?

Для этого стоит взглянуть на следующую загадку:

Время выполнения одной операции в центральном процессоре 1 микросекунда (0,000001 с)

Время выполнения операции внешним устройством 10 миллисекунд (0,010000 с)

Что делать процессору, пока выполняется устройство выполняет операцию?

Архитектура вычислительных машин

Ответ: Заниматься другими задачами и ждать аппаратного прерывания после завершения операции

Именно этой чертой обладали ОС третьего и четвёртого поколений (многозадачность, разделение времени и т.д.)

В сами устройства этих поколений могли похвастать наличием **интеллектуальных контроллеров** внешних устройств. Это позволило разгрузить центральный процессор

Контроллер можно рассматривать как специализированный процессор, управляющий работой "вверенного ему" внешнего устройства по специальным встроенным программам обмена. Такой процессор имеет собственную систему команд. Результаты выполнения каждой операции заносятся во внутренние регистры памяти контроллера, которые могут быть в дальнейшем прочитаны центральным процессором накопителя

Шинная архитектура

Но как обеспечить их взаимодействие между собой? Вот тут и появляется понятие **общей шины (**англ. bus, часто ее называют **магистралью**)

Шина – это несколько проводников, соединяющих несколько устройств. Шины можно разделить на категории в соответствии с выполняемыми функциями. Они могут быть **внутренними** по отношению к процессору и служить для передачи данных в АЛУ и из АЛУ, а могут быть **внешними** по отношению к процессору и связывать процессор с памятью или устройствами ввода-вывода

Каждый тип шины обладает определенными свойствами и к каждому из них предъявляются определенные требования

Шинная архитектура

Шина состоит из трех частей:

- **шина данных**, по которой передается информация
- **шина адреса**, определяющая, кому передаются данные
- **шина управления**, регулирующая процесс обмена информацией

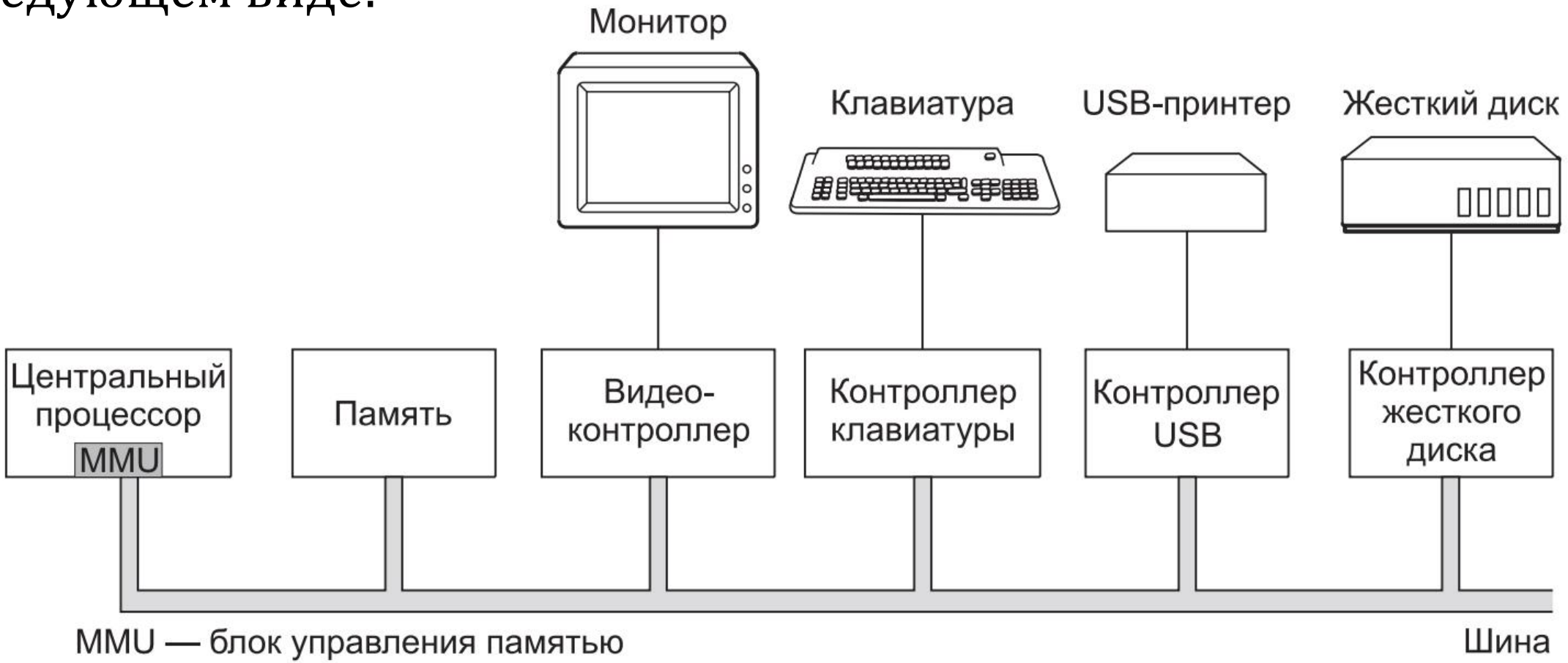
Также может присутствовать **шина питания**

В упрощенном мире можно говорить о том, что центральный процессор, память и устройства ввода-вывода соединены **системной шиной**, по которой они обмениваются информацией друг с другом

Современные же персональные компьютеры имеют более сложную структуру и используют несколько шин

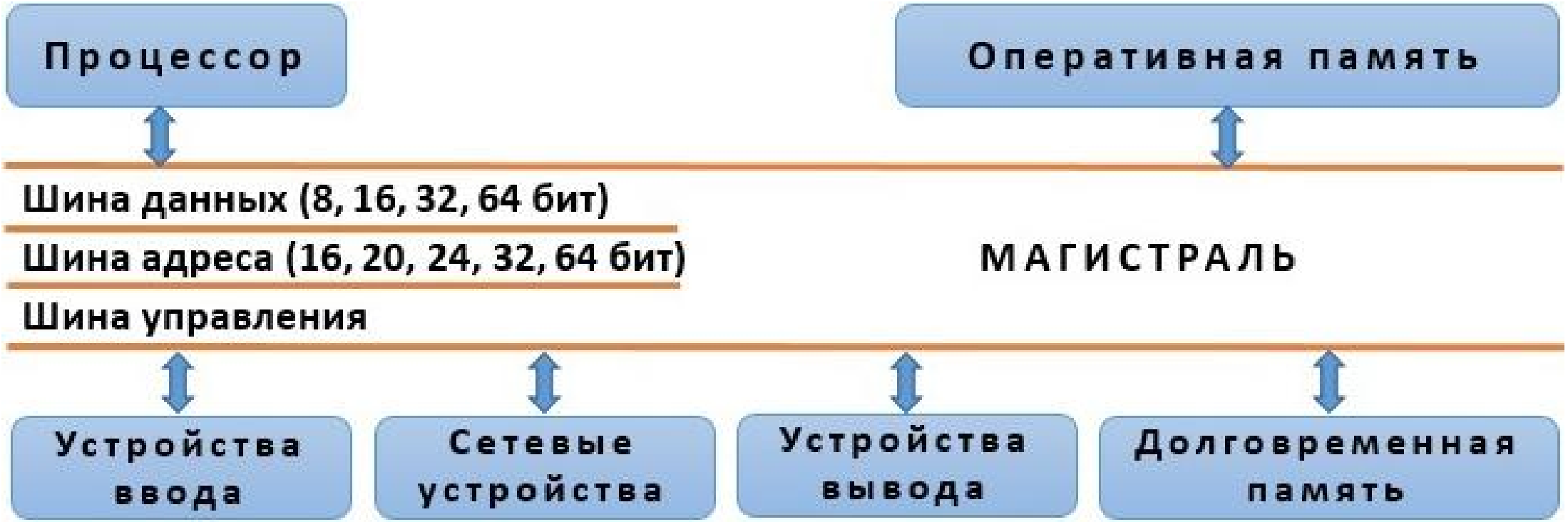
Шинная архитектура

Концептуально простой компьютер можно представить в следующем виде:



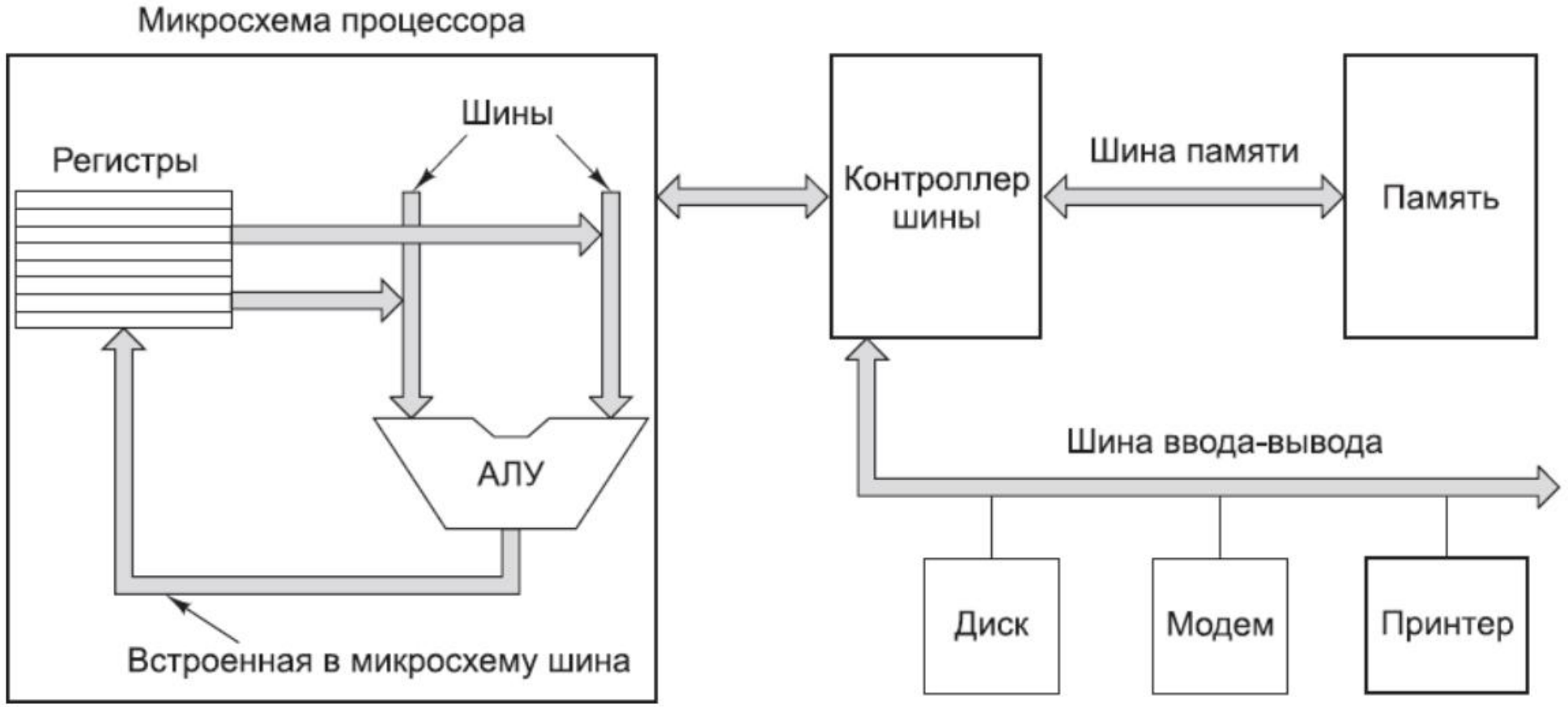
Шинная архитектура

Говоря более подробно о структуре системной шины можно представить следующее:



Шинная архитектура

Как уже было сказано современные устройства используют множество шин:



Шина данных

По этой шине данные передаются между различными устройствами. Разрядность шины данных определяется разрядностью процессора, т. е. количеством двоичных разрядов, которые процессор обрабатывает за один такт

Со времени создания первого персонального компьютера разрядность шины данных увеличилась с 8 до 64 бит

Шина адреса

Каждая ячейка оперативной памяти имеет свой адрес. Адрес передается по адресной шине. Разрядность шины адреса определяет адресное пространство процессора, т. е. количество ячеек оперативной памяти, которые могут иметь уникальные адреса. Количество адресуемых ячеек памяти равно 2^n, где n – разрядность шины адреса

В первых персональных компьютерах разрядность шины адреса составляла 16 бит, в современных персональных компьютерах разрядность шины адреса составляет 64 бита

Но реально в пользовательских устройствах зачастую используются лишь 48 бит в виду достаточности адресного пространства при такой разрядности шины

Шина управления

По шине управления передаются сигналы, определяющие характер обмена информацией по магистрали

Сигналы управления определяют, какую операцию – считывание или запись информации из памяти – нужно производить, синхронизируют обмен информацией между устройствами и т. д.

Устройство, управляющее магистралью – центральное процессорное устройство (ЦПУ) или, проще говоря, процессор

Системная шина

Некоторые устройства, соединенные с шиной, являются **активными** и могут инициировать передачу информации по шине, тогда как другие являются **пассивными** и ждут запросов. Активное устройство называется **задающим**, пассивное – **подчиненным**

Роли могут меняться в зависимости от задач. Например, когда центральный процессор требует от контроллера диска считать или записать блок информации, центральный процессор действует как задающее устройство, а контроллер диска – как подчиненное. Контроллер диска может действовать как задающее устройство, когда он командует памяти принять данные, которые считал с диска

Открытая архитектура

Описанную схему легко пополнять новыми устройствами – это свойство называют **открытостью архитектуры**. Для пользователя открытая архитектура означает возможность свободно выбирать состав внешних устройств для своего компьютера, т.е. конфигурировать его в зависимости от круга решаемых задач

Так как процессор теперь перестал быть центром конструкции, стало возможным реализовывать прямые связи между устройствами ЭВМ. На практике чаще всего используют передачу данных из внешних устройств в ОЗУ и наоборот (**прямой доступ к памяти)**

Открытая архитектура

На практике структура с единственной общей шиной применяется только для ЭВМ с небольшим количеством внешних устройств

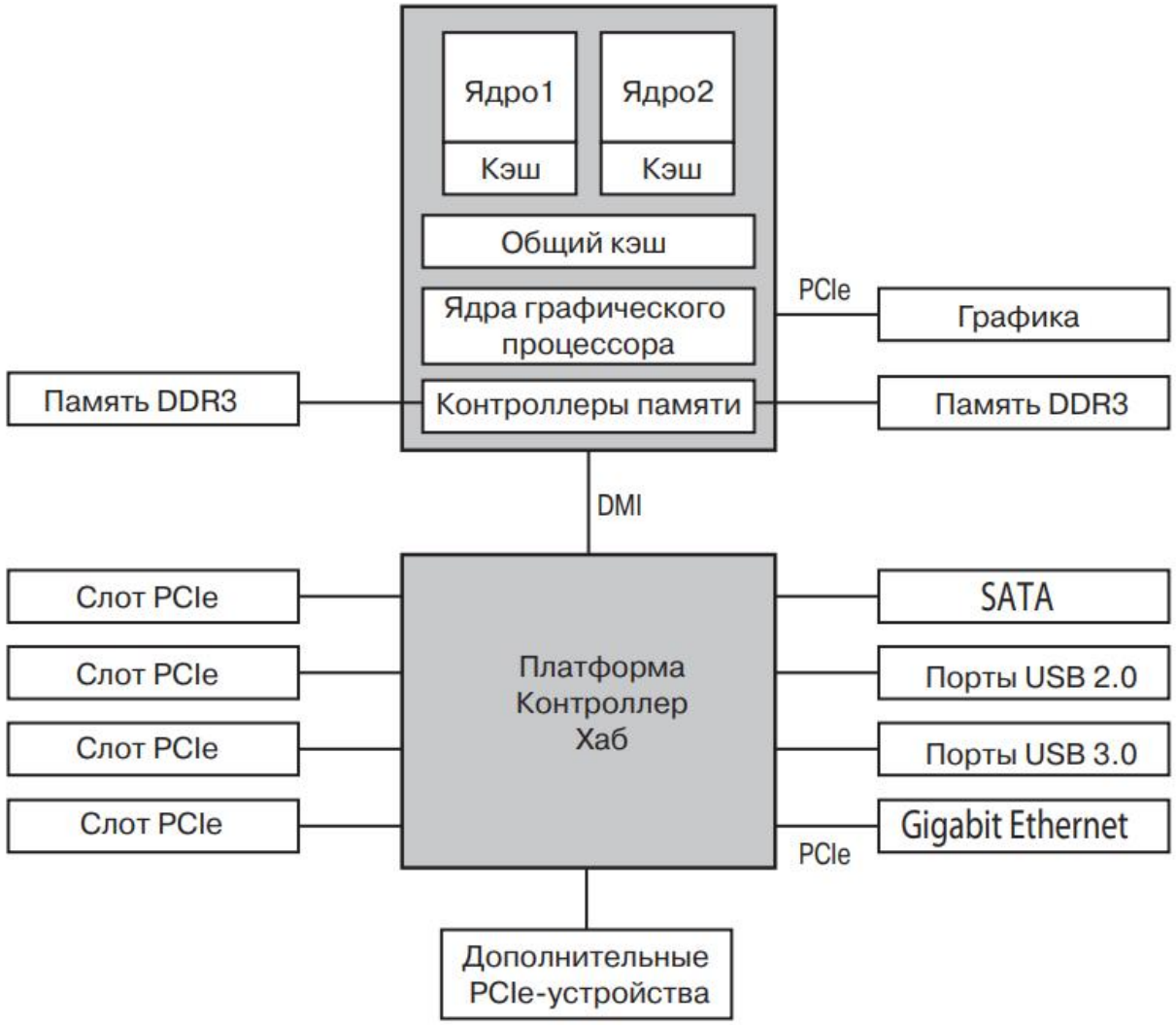
При увеличении потоков информации между устройствами ЭВМ единственная магистраль перегружается, что существенно тормозит работу компьютера

Поэтому в состав ЭВМ могут вводиться одна или несколько дополнительных шин. Например, одна шина может использоваться для обмена с памятью, вторая – для связи с «быстрыми», а третья – с «медленными» внешними устройствами. Отметим, что высокоскоростная шина данных ОЗУ обязательно требуется при наличии режима ПДП

Открытая архитектура

Вследствие этой эволюции массовая x86-система на данный момент имеет следующий вид

У этой системы имеется множество шин (например, шина кэш-памяти, шина памяти, а также шины PCIe, PCI, USB, SATA и DMI), каждая из которых имеет свою скорость передачи данных и свое предназначение. Операционная система для осуществления функций настройки и управления должна знать обо всех этих шинах



Модульно-магистральная архитектура

Подводя краткие итоги об архитектуре вычислительных машин, хотелось бы отметить что в современных терминах описанная ранее архитектура взаимодействия называется **модульно-магистральной архитектурой**

Также стоит закрепить тот факт, что главной причиной усложнения архитектуры вычислительных машин стало желание **как можно эффективнее использовать процессор**

Примеры **внутренних** компьютерных шин: ISA, PCI

Примеры **внешних** компьютерных шин: IDE, USB, SCSI, SATA

Архитектура вычислительных машин

Также хотелось бы упомянуть две существующие архитектуры построения ЭВМ, а именно:

- **Архитектура фон Неймана**
- **Гарвардская архитектура**

Говоря о современных устройствах первый вариант чаще встречается в персональных устройствах, а второй – в контроллерах и микроконтроллерах, а также, например, в реализации кэш-памяти процессора

Архитектура вычислительных машин

Классическая архитектура ЭВМ, построенная по принципу фон Неймана и реализованная в вычислительных машинах двух (трех) поколений, содержит следующие основные блоки:

- арифметико-логическое устройство (АЛУ), выполняющее арифметические и логические операции
- управляющее устройство (УУ), организующее процесс выполнения программ
- внешнее запоминающее устройство (ВЗУ), или память, для хранения программ и данных
- оперативное запоминающее устройство (ОЗУ)
- устройства ввода и вывода информации (УВВ)

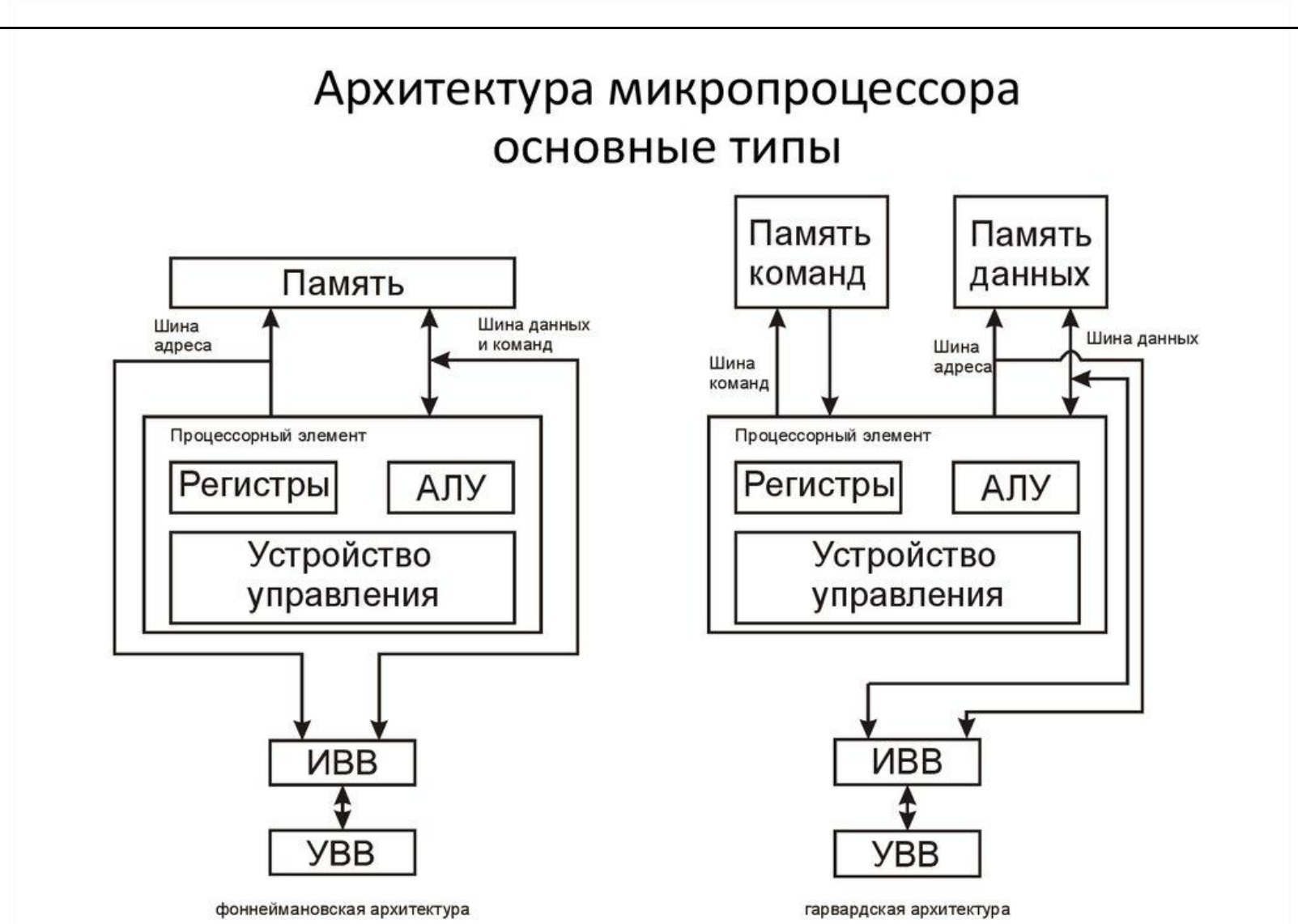
Архитектура вычислительных машин

В целом гарвардская архитектура ничем не отличается от фон Неймановской, за исключением принципа хранения данных и программ, а именно в этом случае они хранятся в различных запоминающих устройствах

Также из этого следует, что каналы данных и инструкций также разделены в виде отдельных шин

В компьютере с использованием гарвардской архитектуры процессор может считывать очередную команду и оперировать памятью данных одновременно и без использования кэш-памяти, что невозможно в архитектуре фон Неймана

Архитектура вычислительных машин



Организация работы компьютера

Как уже говорилось, с точки зрения программиста, операционная система – это программа, добавляющая ряд команд и функций к командам и функциям, выполняемым аппаратным обеспечением

Три важные группы таких функций:

- **Виртуальная память**. Механизм виртуальной памяти используется многими операционными системами. Она позволяет создать впечатление, будто у машины больше памяти, чем есть на самом деле
- **Файловый ввод-вывод**. Это понятие более высокого уровня, чем команды ввода-вывода
- **Параллелизм** (как организовано одновременное выполнение нескольких процессов, обмен информацией и синхронизация). Понятие процесса является очень важным, и мы подробно рассмотрим его в следующих лекциях

Организация работы компьютера

Если рассматривать более подробно, то самыми важными задачами управления компьютерным оборудованием, осуществляемыми операционной системой, являются следующие:

- Параллельное функционирование модулей ввода, вывода информации и процессора
- Организация кэширования данных и выполнение согласования скоростей информационного обмена
- Разбиение модулей и информационных данных среди процессов
- Организация удобной работы логического интерфейса между модулями и оставшейся частью системы
- Организация поддержки различных устройств с обеспечением возможности просто их добавить
- Режим динамической загрузки и выгрузки драйверов
- Обеспечение поддержки набора файловых систем

Организация работы компьютера

Модули, предназначенные для ввода и вывода информации, подразделяются на следующие типы:

- Модули, ориентированные на работу с **блоками**
- Модули, ориентированные на работу с **байтами**

Ориентированное на работу с блоками оборудование сохраняет данные в блоках, имеющих фиксированный размер и свой уникальный адрес. Примером такого устройства может служить жёсткий диск

Модули, работающие с байтами, не имеют адресации и не обеспечивают возможность поиска информации. Они только могут генерировать или потреблять байтовую очерёдность. В качестве примера таких устройств можно привести сетевые адаптеры, терминалы и так далее

Организация работы компьютера

Основные принципы построения ПО ввода и вывода информации:

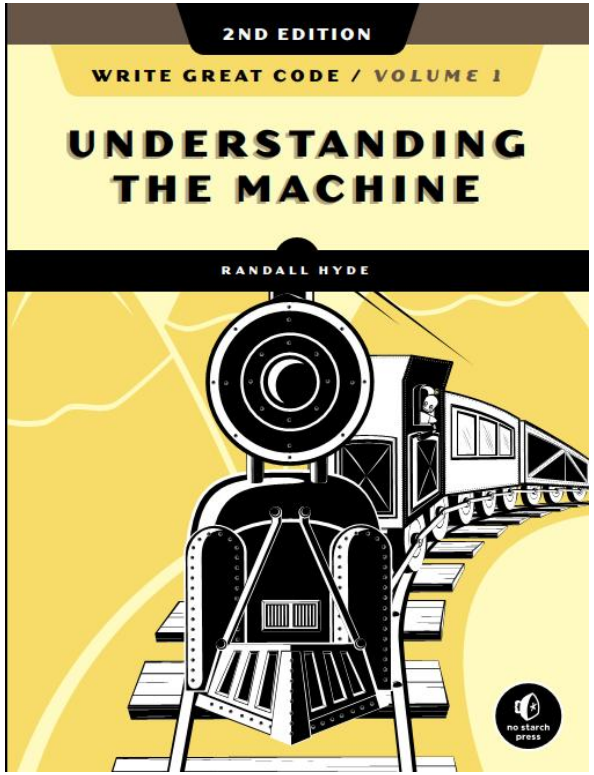
- Разбиение на отдельные уровни
- Обработка ошибок

Программное обеспечение ввода и вывода информации делится на следующие уровни:

- Уровень обработки прерываний
- Уровень драйверов оборудования
- Уровень независимого от оборудования слоя операционной системы (системные вызовы)
- Пользовательский уровень программного обеспечения (функции библиотек)

Совет

Для закрепления знаний по вопросам устройства ЭВМ есть замечательная книга:



Операционные системы

Лекция 2

Архитектура вычислительных машин

Запуск ОС

В кратком изложении загрузка компьютера происходит следующим образом:

У каждого персонального компьютера есть материнская плата. На материнской плате находится программа, которая называется базовой системой ввода-вывода – **BIOS** (Basic Input Output System)

BIOS содержит низкоуровневое программное обеспечение ввода-вывода, включая процедуры считывания состояния клавиатуры, вывода информации на экран и осуществления, ко всему прочему, дискового ввода-вывода

В наши дни эта программа хранится в энергонезависимой флеш-памяти с произвольным доступом, которая может быть обновлена операционной системой в случае обнаружения в BIOS различных ошибок

Запуск ОС

При начальной загрузке компьютера BIOS начинает работать первой. Сначала она проверяет объем установленной на компьютере оперативной памяти и наличие клавиатуры, а также установку и нормальную реакцию других основных устройств

Все начинается со сканирования шин PCIe и PCI с целью определения всех подключенных к ним устройств. Некоторые из этих устройств унаследованы из прошлого

Они имеют фиксированные уровни прерываний и адреса ввода-вывода (возможно, установленные с помощью переключателей или перемычек на карте ввода-вывода, но не подлежащие изменению со стороны операционной системы). Эти устройства регистрируются

Устройства, отвечающие стандарту plug and play, также регистрируются. Если присутствующие устройства отличаются от тех, которые были зарегистрированы в системе при ее последней загрузке, то производится конфигурирование новых устройств

Запуск ОС

Затем BIOS определяет устройство, с которого будет вестись загрузка, по очереди проверив устройства из списка, сохраненного в CMOS-памяти. Пользователь может внести в этот список изменения, войдя сразу после начальной загрузки в программу конфигурации BIOS. Обычно делается попытка загрузки с компакт-диска (иногда с флеш-накопителя USB), если, конечно, таковой присутствует в системе

В случае неудачи система загружается с жесткого диска. С загрузочного устройства в память считывается первый сектор, а затем выполняется записанная в нем программа. Обычно эта программа проверяет таблицу разделов, которая находится в конце загрузочного сектора, чтобы определить, какой из разделов имеет статус активного

Затем из этого раздела считывается вторичный загрузчик, который в свою очередь считывает из активного раздела и запускает операционную систему

Запуск ОС

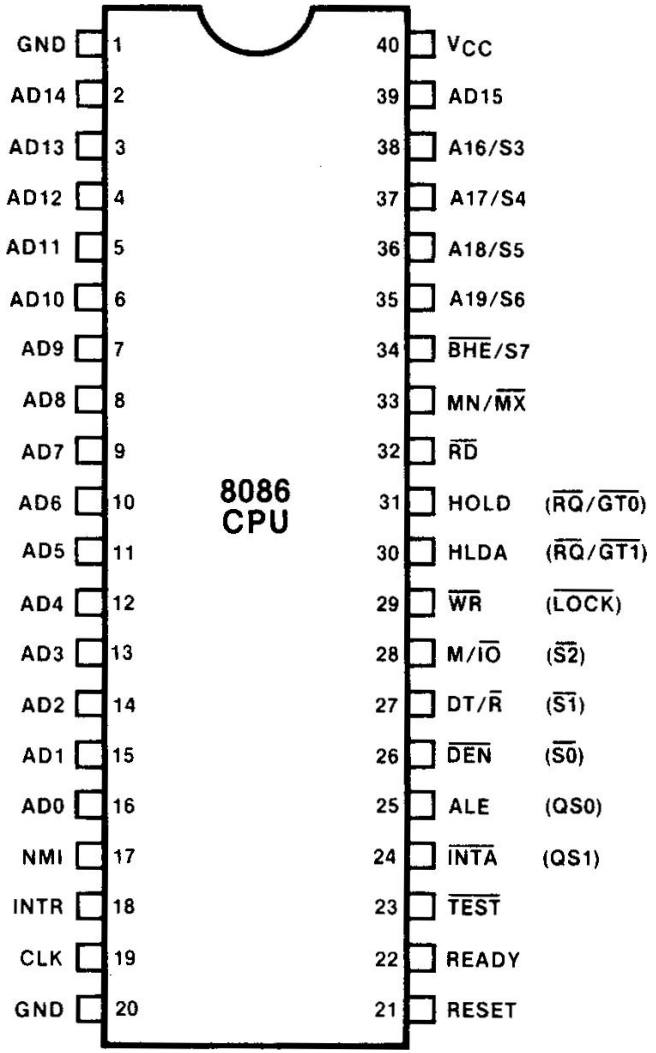
После этого операционная система запрашивает BIOS, чтобы получить информацию о конфигурации компьютера. Она проверяет наличие драйвера для каждого устройства.

Если драйвер отсутствует, операционная система просит установить компакт-диск с драйвером (поставляемый производителем устройства) или загружает драйвер из Интернета. Как только в ее распоряжении окажутся все драйверы устройств, операционная система загружает их в ядро

Затем она инициализирует свои таблицы, создает все необходимые ей фоновые процессы и запускает программу входа в систему или графический интерфейс пользователя

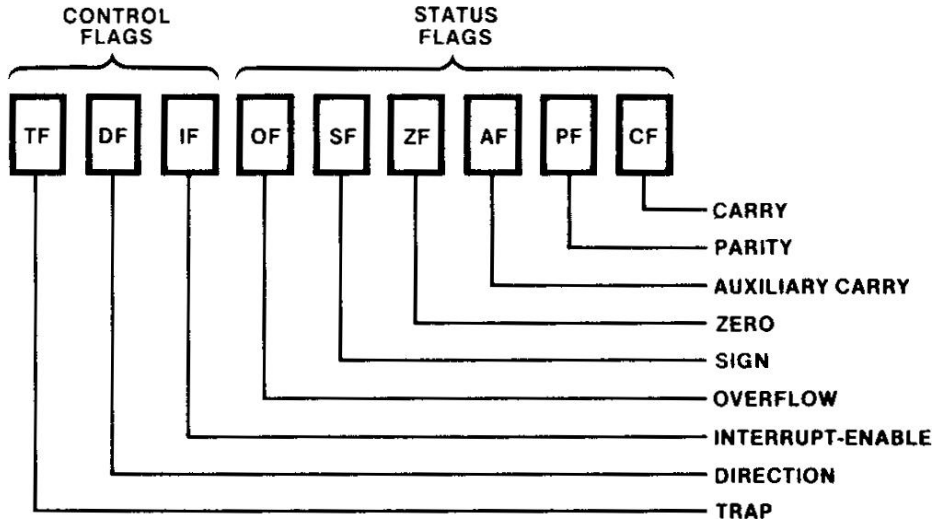
Особенности аппаратной реализации в микропроцессоре 8086

- *Два десятка последующих слайдов не нужно заучивать. Они призваны лишь нарисовать общую картину*
- Реализацию аппаратных прерываний рассмотрим на примере одного из простейших процессоров — основоположнике целой серии x86 — на примере микропроцессора 8086
- Нас интересуют только следующие выводы:
- **NMI** (Non maskable interrupt) – немаскируемое прерывание
- **INTR** (Interrupt Request) – внешнее (маскируемое) прерывание
- **INTA** (Interrupt acknowledge)
- **M/$\overline{IO}$** – переключает операции с внешней памятью (= 1) и с пространством ввода-вывода (= 0)

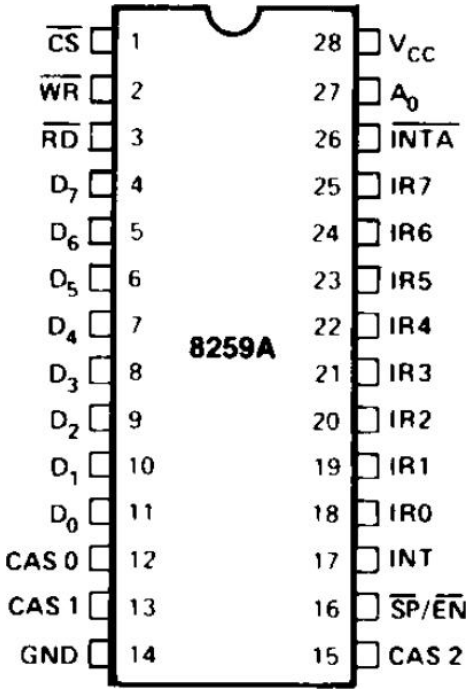


Регистры

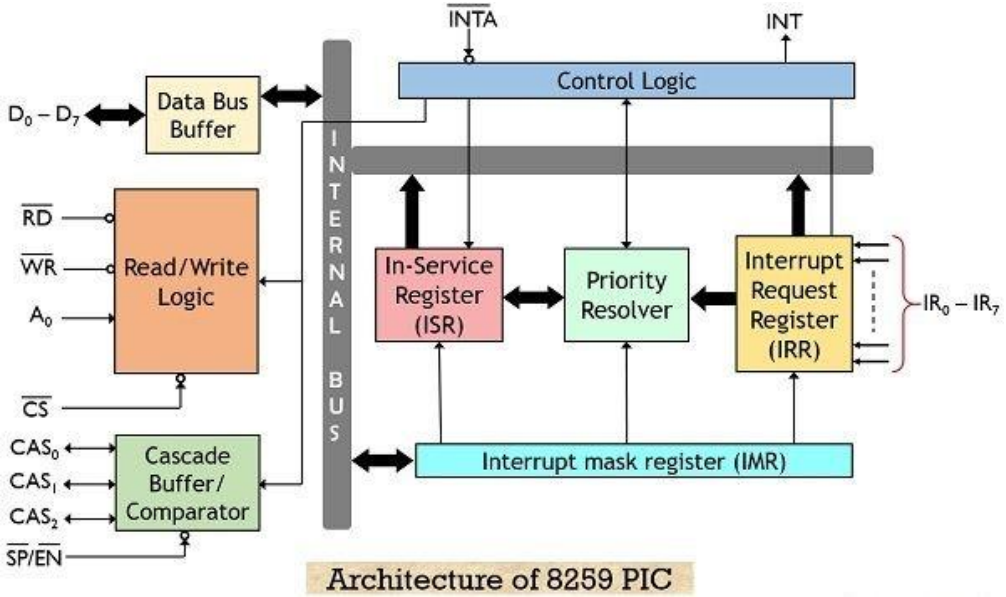
- **Регистр флагов**
- Установка флага IF (the interrupt-enable flag) разрешает процессору распознавать внешние (маскируемые) запросы на прерывание. Флаг не влиеят на немаскируемые и внутренне генерируемые прерывания.
- Регистр указателя инструкции **IP** (Instruction Pointer)
- Регистр указателя стека **SP** (Stack Pointer)
- **Сегментные регистры** (CS, SS, DS, ES…)



Микроконтроллер 8259A



- Программируемый микроконтроллер 8259A принимает сигналы прерываний от внешних устройств и отправляет самый приоритетный из них на микропроцессор. Номер вектора передается по контактам D_0-D_7.
- Позволяет избирательно маскировать отдельные источники прерываний.
- Допускает каскадирование.



Architecture of 8259 PIC

Пример обработки прерывания для x86 в реальном режиме (1 из 10)

- Состояние процессора до прерывания, в данный момент времени выполняется команда по адресу 1C20:3617

- *Примечание*.
- *Реальный адрес команды вычисляется как IP + CS * 16:*
- **1C200**
  +
  3617
- **1F817**

Регистры процессора	
IP	**3618**
CS	1C20
Flags	0020 (IF=1)
SP	**03FA**

Сегмент 1C20	
	. . .
3617	Текущая команда
3618	**Следующая команда**
361B	. . .
	. . .

Сегмент 0AF0	
0458	push ax
0459	…
0461	sti
	. . .
0498	cli
	…
04B5	pop ax
04B6	iret

Таблица векторов прерываний		
0000	12F4	Прерывание 0
0002	0AF0	
0004	0458	Прерывание 1
0006	0AF0	
0008	0540	Прерывание 2
000a	0AF0	
000c	0770	Прерывание 3
000e	0CC8	
0010	. . .	

Стек потока		
03E8		
03EA		
03EC		
03EE		
03F0		
03F2		
03F4		
03F6		
03F8		
03FA		Уже занятое пространство
03FC		
03FE		

Пример обработки прерывания для x86 в реальном режиме (2 из 10)

- Происходит прерывание с номером 2, работа текущей команды продолжается до ее завершения

Регистры процессора	
IP	**3618**
CS	1C20
Flags	0020 (IF=1)
SP	03FA

Сегмент 1C20	
	. . .
3617	Текущая команда
3618	**Следующая команда**
361B	. . .
	. . .

Сегмент 0AF0	
0458	push ax
0459	…
0461	sti
	. . .
0498	cli
	…
04B5	pop ax
04B6	iret

Таблица векторов прерываний		
0000	12F4	Прерывание 0
0002	0AF0	
0004	0314	Прерывание 1
0006	0AF0	
0008	0458	Прерывание 2
000a	0AF0	
000c	0770	Прерывание 3
000e	0CC8	
0010	. . .	

Стек потока		
03E8		
03EA		
03EC		
03EE		
03F0		
03F2		
03F4		
03F6		
03F8		
03FA		Уже занятое пространство
03FC		
03FE		

Пример обработки прерывания для x86 в реальном режиме (3 из 10)

- Текущее состояние сохраняется в стек
- Процессор помнит, что номер прерывания равен 2

Регистры процессора	
IP	3618
CS	1C20
Flags	0000 (IF=0)
SP	03F4

Сегмент 1C20	
	. . .
3617	Выполненная команда
3618	Следующая команда
361B	. . .
	. . .

Сегмент 0AF0	
0458	push ax
0459	…
0461	sti
	. . .
0498	cli
	…
04B5	pop ax
04B6	iret

Таблица векторов прерываний		
0000	12F4	Прерывание 0
0002	0AF0	
0004	0314	Прерывание 1
0006	0AF0	
0008	0458	Прерывание 2
000a	0AF0	
000c	0770	Прерывание 3
000e	0CC8	
0010	. . .	

Стек потока		
03E8		
03EA		
03EC		
03EE		
03F0		
03F2		
03F4	3618	Старый IP
03F6	1C20	Старый CS
03F8	0020 (IF=1)	Старый Flags
03FA		Уже занятое пространство
03FC		
03FE		

Пример обработки прерывания для x86 в реальном режиме (4 из 10)

- Новые значения регистров загружаются из таблицы векторов прерываний
- Каждый вектор занимает в памяти 4 байта.
- Так как номер прерывания был 2, смещение в таблице векторов прерываний будет 8

Регистры процессора	
IP	**0458**
CS	**0AF0**
Flags	0000 (IF=0)
SP	03F4

Сегмент 1C20	
	. . .
3617	Выполненная команда
3618	Следующая команда
361B	. . .
	. . .

Сегмент 0AF0	
0458	push ax
0459	…
0461	sti
	. . .
0498	cli
	…
04B5	pop ax
04B6	iret

2 x 4 = 8

Таблица векторов прерываний		
0000	12F4	Прерывание 0
0002	0AF0	
0004	0314	Прерывание 1
0006	0AF0	
0008	**0458**	**Прерывание 2**
000a	**0AF0**	
000c	0770	Прерывание 3
000e	0CC8	
0010	. . .	

Стек потока		
03E8		
03EA		
03EC		
03EE		
03F0		
03F2		
03F4	3618	Старый IP
03F6	1C20	Старый CS
03F8	0020 (IF=1)	Старый Flags
03FA		Уже занятое пространство
03FC		
03FE		

Пример обработки прерывания для x86 в реальном режиме (5 из 10)

- Обработчик прерывания готов к выполнению

Регистры процессора	
IP	0459
CS	0AF0
Flags	0000 (IF=0)
SP	03F4

Сегмент 1C20	
	. . .
3617	Выполненная команда
3618	Следующая команда
361B	. . .
	. . .

Сегмент 0AF0	
0458	push ax
0459	…
0461	sti
	. . .
0498	cli
	…
04B5	pop ax
04B6	iret

Таблица векторов прерываний		
0000	12F4	Прерывание 0
0002	0AF0	
0004	0314	Прерывание 1
0006	0AF0	
0008	0458	Прерывание 2
000a	0AF0	
000c	0770	Прерывание 3
000e	0CC8	
0010	. . .	

Стек потока		
03E8		
03EA		
03EC		
03EE		
03F0		
03F2		
03F4	3618	Старый IP
03F6	1C20	Старый CS
03F8	0020 (IF=1)	Старый Flags
03FA		Уже занятое пространство
03FC		
03FE		

Пример обработки прерывания для x86 в реальном режиме (6 из 10)

- Выполнена первая команда обработчика прерывания
- В данном случае это сохранение регистра AX в стек
- Предполагается, что этот регистр обработчик будет использовать в своей работе

Регистры процессора	
IP	**045B**
CS	0AF0
Flags	0000 (IF=0)
SP	**03F2**

Сегмент 1C20	
	. . .
3617	Выполненная команда
3618	Следующая команда
361B	. . .
	. . .

Сегмент 0AF0	
0458	push ax
0459	...
0461	sti
0462	. . .
0498	cli
	...
04B5	pop ax
04B6	iret

Таблица векторов прерываний		
0000	12F4	Прерывание 0
0002	0AF0	
0004	0314	Прерывание 1
0006	0AF0	
0008	0458	Прерывание 2
000a	0AF0	
000c	0770	Прерывание 3
000e	0CC8	
0010	. . .	

Стек потока		
03E8		
03EA		
03EC		
03EE		
03F0		
03F2	**0005**	Старый AX (не показано)
03F4	3618	Старый IP
03F6	1C20	Старый CS
03F8	0020 (IF=1)	Старый Flags
03FA		Уже занятое пространство
03FC		
03FE		

Пример обработки прерывания для x86 в реальном режиме (7 из 10)

- После завершения критической части обработчика прерывания выполнена команда **`sti`**, после чего может быть продолжена обычная работа обработчика
- В это время могут возникать более приоритетные прерывания
- Благодаря стеку будет выполнен корректный выход из каждого обработчика

Регистры процессора	
IP	0462
CS	0AF0
Flags	0020 (IF=1)
SP	03F2

Сегмент 1C20	
	. . .
3617	Выполненная команда
3618	Следующая команда
361B	. . .
	. . .

Сегмент 0AF0	
0458	push ax
0459	...
0461	sti
0462	. . .
0498	cli
	...
04B5	pop ax
04B6	iret

Таблица векторов прерываний		
0000	12F4	Прерывание 0
0002	0AF0	
0004	0314	Прерывание 1
0006	0AF0	
0008	0458	Прерывание 2
000a	0AF0	
000c	0770	Прерывание 3
000e	0CC8	
0010	. . .	

Стек потока		
03E8		
03EA		
03EC		
03EE		
03F0		
03F2	0005	Старый AX (не показано)
03F4	3618	Старый IP
03F6	1C20	Старый CS
03F8	0020 (IF=1)	Старый Flags
03FA		Уже занятое пространство
03FC		
03FE		

Пример обработки прерывания для x86 в реальном режиме (8 из 10)

- Выполнена команда **`cli`** перед еще одним критическим участком (в данном случае восстановлением регистров), чтобы предотвратить прерывание этого процесса и возможное разрушение данных
- Флаг разрешения прерывания сброшен в ноль

Регистры процессора	
IP	0499
CS	0AF0
Flags	0000 (IF=0)
SP	03F2

Сегмент 1C20	
	. . .
3617	Выполненная команда
3618	Следующая команда
361B	. . .
	. . .

Сегмент 0AF0	
0458	push ax
0459	. . .
0461	sti
0462	. . .
0498	cli
0499	…
04B5	pop ax
04B6	iret

Таблица векторов прерываний		
0000	12F4	Прерывание 0
0002	0AF0	
0004	0314	Прерывание 1
0006	0AF0	
0008	0458	Прерывание 2
000a	0AF0	
000c	0770	Прерывание 3
000e	0CC8	
0010	. . .	

Стек потока		
03E8		
03EA		
03EC		
03EE		
03F0		
03F2	0005	Старый AX (не показано)
03F4	3618	Старый IP
03F6	1C20	Старый CS
03F8	0020 (IF=1)	Старый Flags
03FA		Уже занятое пространство
03FC		
03FE		

Пример обработки прерывания для x86 в реальном режиме (9 из 10)

- Восстановлен регистр AX
- Указатель стека увеличивается на 2, участок памяти, который ранее хранил старое значение AX, по прежнему его содержит, но на это нельзя полагаться — любые операции со стеком могут его изменить

Регистры процессора	
IP	**04B7**
CS	0AF0
Flags	0000 (IF=0)
SP	**03F4**

Сегмент 1C20	
	. . .
3617	Выполненная команда
3618	Следующая команда
361B	. . .
	. . .

Сегмент 0AF0	
0458	push ax
0459	. . .
0461	sti
0462	. . .
0498	cli
0499	…
04B5	pop ax
04B6	iret

Таблица векторов прерываний		
0000	12F4	Прерывание 0
0002	0AF0	
0004	0314	Прерывание 1
0006	0AF0	
0008	0458	Прерывание 2
000a	0AF0	
000c	0770	Прерывание 3
000e	0CC8	
0010	. . .	

Стек потока		
03E8		
03EA		
03EC		
03EE		
03F0		
03F2	0005	"Мусор"
03F4	3618	Старый IP
03F6	1C20	Старый CS
03F8	0020 (IF=1)	Старый Flags
03FA		Уже занятое пространство
03FC		
03FE		

Пример обработки прерывания для x86 в реальном режиме (10 из 10)

- Восстанавливается состояние потока, прерванная программа готова к выполнению

Регистры процессора	
IP	3618
CS	1C20
Flags	0020 (IF=1)
SP	03FA

Сегмент 1C20	
	. . .
3617	Выполненная команда
3618	Следующая команда
361B	. . .
	. . .

Сегмент 0AF0	
0458	push ax
0459	. . .
0461	sti
0462	. . .
0498	cli
0499	...
04B5	pop ax
04B6	iret

Таблица векторов прерываний		
0000	12F4	Прерывание 0
0002	0AF0	
0004	0314	Прерывание 1
0006	0AF0	
0008	0458	Прерывание 2
000a	0AF0	
000c	0770	Прерывание 3
000e	0CC8	
0010	. . .	

Стек потока		
03E8		
03EA		
03EC		
03EE		
03F0		
03F2	0005	"Мусор"
03F4	3618	"Мусор"
03F6	1C20	"Мусор"
03F8	0020 (IF=1)	"Мусор"
03FA		Уже занятое пространство
03FC		
03FE		