# Тема 2. Теоретическая часть

В предыдущей теме было рассмотрено множество моделей касающихся как ОС, так и компьютера в целом. В частности, было утверждение, что *ОС является некоторой виртуальной машиной*, которая установлена на аппаратных средствах некоторой ЭВМ.

Большинство современных ОС могут работать на аппаратных средствах ЭВМ различных архитектур, при этом, пользователь может и не знать об этих различиях, поскольку внешние проявления в поведении таких ОС могут быть неразличимы. Достигается это специальной архитектурой ядра ОС, в которой:

* *стандартизируется интерфейс* (API) между режимами ядра и пользователя;
* *выделяется программная прослойка* между ядром ОС и аппаратным обеспе- чением ЭВМ, реализуемая с помощью *модулей (драйверов)*, которые или ста-

тически компилируются с ядром или загружаются по мере необходимости.

С другой стороны, прежде чем ОС начнет функционировать, ее ядро должно быть загружено в память ЭВМ. Для этого служит специальное ПО, которое входит в состав самого компьютера и является его неотъемлемой частью, независимо от установленной ОС.

**В данной теме**, мы рассмотрим такое ПО, которое появилось с развитием средств микропроцессорной техники, позволившей заменить специальные аппаратные средства загрузки ОС на программные.

**По традиции**, изложение учебного материала будет дано *для архитектуры процес- сора x86*, модификации которого установлены в компьютерных классах кафедры АСУ и широко используются в переностных ЭВМ.

## Архитектура х86

Чтобы выйти на должный уровень понимания, следует рассмотреть ряд воп- росов, касающихся аппаратной части компьютеров. Традиционно, в качестве основ- ной модели компьютера, рассматриваются изделия созданные на базе процессора ***x86***.

**x86** (***Intel 80x86***) — архитектура процессора с одноименным набором команд, ко- торая впервые была реализованна в процессорах компании *Intel* и относится к серии ее процессоров ранних моделей — ***80086, 80386 (i386), 80486 (i486)***.

На рисунке 1.1, показаны *16-битные регистры* процессора ***80086***, обеспечи- вающие работу компьютера на *20-битной шине адреса*:

* *для выполнения арифметических и логических операций* служат регистры об- щего назначения: AX, BX, CX и DX;
* *индексные регистры* служат для формирования массивов; для этих же целей служат *указательные регистры*;
* *регистр состояния* содержит биты, которые изменяются в процессе выпол- нения различных операций;
* *сегментные регистры* являются указателями на начало областей оператив- ной памяти (сегменты);
* *указатель команды* — смещение относительно начала сегмента команд, определяемого регистром CS.

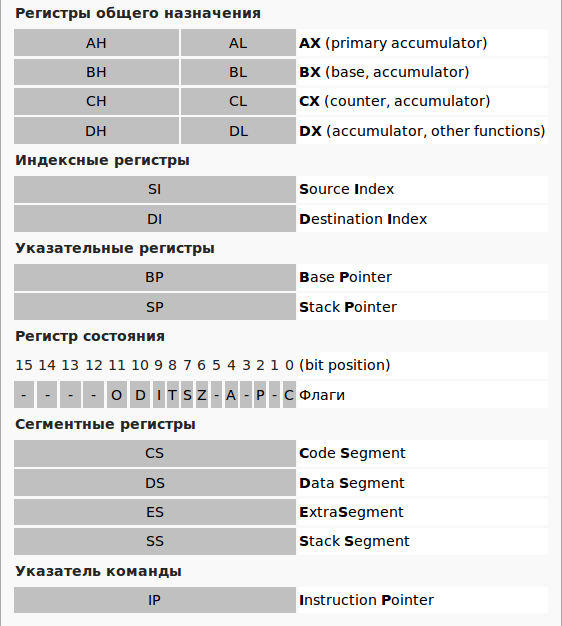


Рисунок 1.1 - Регистры процессора 80086

**В общем случае**, для работы с регистрами процессора необходимо знать набор его команд, которые на практике пишутся на языке ассемблера.

*Поскольку изучение ассемблера* не входит в программу нашего обучения, то мы от- метим основные качественные характеристики этого процессора:

* объем адресуемой памяти *1 Мбайт*;
* работа в реальном режиме: *защищенный режим работы отсутствует*.

**Преемником** компании ***Intel***, для дешевых процессоров x86, стала компания ***AMD***.

*Со временем*, появились ***32-битные*** и ***64-битные*** процессора.

*Начиная с процессора 80386*, появился полноценный защищенный режим работы, который стал обеспечивать ядрам ОС *привелигированный режим работы*.

**Для 64-битных процессоров** и соответствующего ПО, совместимых с набором команд ***х86***, стало применяться обозначение ***х86-64***. На рисунке 1.2, показаны ре- гистры процессора ***AMD64***.

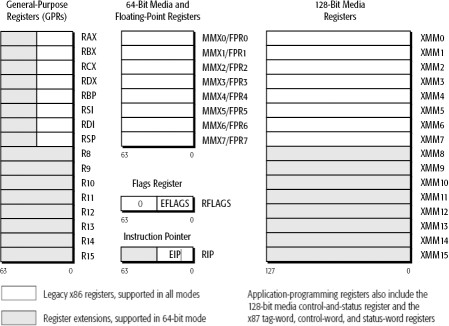


Рисунок 1.2 - Набор регистров процессоров х86-64

**Замечание** Хотя 32-битное ПО может работать на процессорах х86-64, *64-битное ПО не может работать на процессорах х86*. Несмотря на расширение количества регистров и увели- чение их размеров, имеется подмножество команд, которые выполняются на всех процессорах. Такой набор команд соответствует *процессору 80386*, а набор команд часто в дистрибутивах ОС обозначается как *i386*.

На рисунке 1.2 (б) показан полный набор регистров процессовов x86.

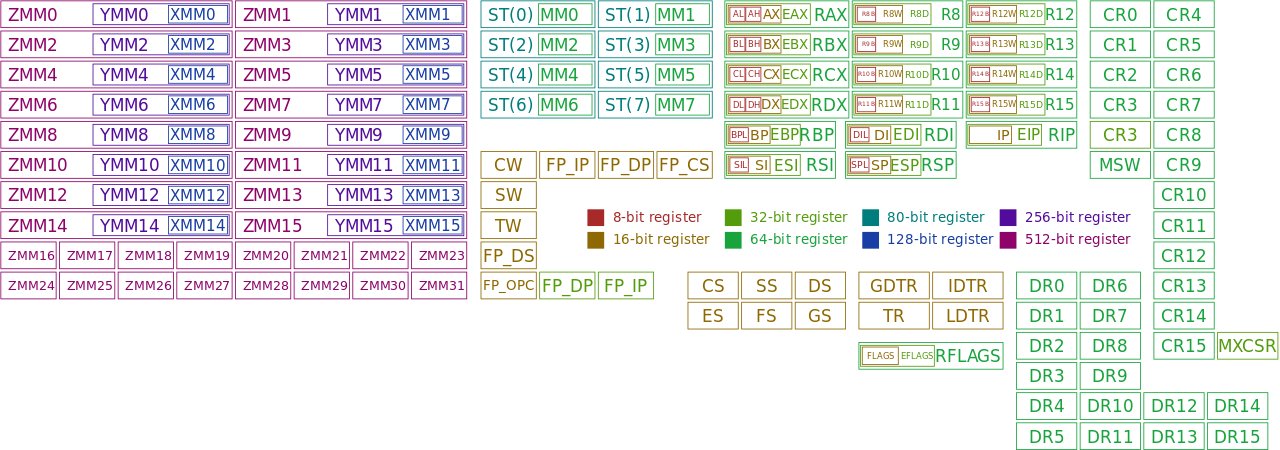


Рисунок 1.2 (б) — Полный набор регистров процессоров x86

**Среди всего множества команд**, которые типично работают на всех процессорах и используют сегмент стека, *выделяются две*:

* *вызов функции* или команда: **CALL адрес**; в стек заносятся сегмент и смеще- ние следующей за вызовом функции команды, а затем управление передается

по адресу указанному в команде CALL; когда функция заканчивает работу, то из стека извлекаются сохраненные сегмент и смещение, которые помещаются в регистры CS и IP, обеспечивая продолжение работы программы;

* *прерывание* или команда: **INTERRUPT номер**; кроме сегмента и смещения команд, в стек заносятся регистр флагов, тем самым сохраняя полное состоя-

ние процессора; **номер** соответствует *номеру слова* в начале памяти ЭВМ, в которые предварительно записаны значения сегмента и смещения, соответст- вующие адресу обработчика прерывания; таким образом обычно обрабатыва- ются аппаратные прерывания процессора.

**Другим важным аспектом работы** аппаратного обеспечения ЭВМ является *спо- соб выполения операций ввода-вывода*. Существует три таких способа:

* ***Программируемый ввод-вывод***, когда процессор посылает команды, связан- ные с ним контроллеру, а затем *периодически проверяет состояние модуля*

*ввода-вывода с целью проверки завершения операции*.

* ***Ввод-вывод, управляемый прерываниями***, когда процессор посылает необхо- димые команды контроллеру ввода-вывода и продолжает выполнять текущий

процесс, если нет необходимости в ожидании выполнения операции ввода- вывода. В ином случае, текущий процесс приостанавливается до получения сигнала прерывания о завершении ввода-вывода, а процессор переключается на выполнение другого процесса. Наличие прерываний процессор проверяет в конце каждого цикла выполняемых команд.

* ***Прямой доступ к памяти (direct memory access – DMA)***. В этом случае, имеется специальный *аппаратный модуль прямого доступа к памяти*, кото- рый управляет обменом данных между основной памятью и контроллером

ввода-вывода. При этом, процессор посылает запрос на передачу блока

данных модулю DMA, а само прерывание происходит только после передачи всего блока данных.

**В современных компьютерах** используется прямой способ доступа к памяти, схема которого показана на рисунке 1.3.

**Подсистема ввода-вывода** должна также учитывать *режим работы шины*, кото- рый может быть *пословным* или *поблочном*.

*В пословном режиме*, контроллер DMA выставляет запрос на перенос одного слова и получает его. Если процессору также нужна эта шина, ему приходится подож- дать.

**Такой механизм** называется ***захватом цикла***, потому, что контроллер устройства периодически забирает случайный цикл шины у центрального процессора, слегка тормозя его.

На рисунке 1.4, показана позиция цикла команд, в которых работа процессора может быть приостановлена. В любом случае, приостановка процессора проис- ходит только при необходимости использования шины. После этого, устройство DMA выполняет передачу слова и возвращает управление процессору.

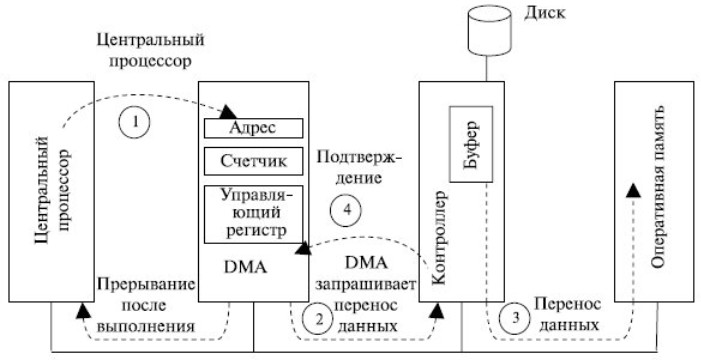


Рисунок 1.3 - Схема прямого доступа к памяти с помошью DMA

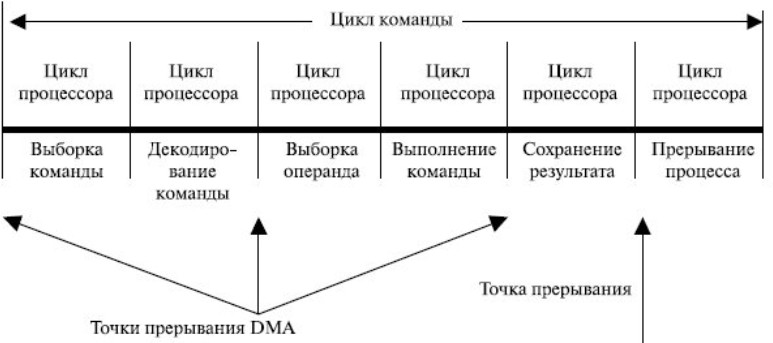


Рисунок 1.4 - Точки прерывания процессора устройством DMA

**В блочном режиме**, работы контроллер DMA занимает шину на серию пересылок (***пакет***). Такой режим более эффективен, однако при переносе большого блока данных центральный процессор и другие устройства могут быть заблокированы на существенный промежуток времени.

**При большом количестве устройств**, от подсистемы ввода-вывода *требуется спланировать свою работу в реальном масштабе времени*, в котором работают все внешние устройства, запуск и приостановку большего количества разных драйве- ров, обеспечив, при этом, время реакции каждого драйвера на независимые собы- тия контролеров внешних устройств.

**С другой стороны**, необходимо минимизировать загрузку процессора задачами ввода-вывода.

**Решение** этих задач достигается на основе *многоуровневой приоритетной схемы обслуживания прерываний*: для обеспечения приемлемого уровня реакции, все драйверы распределяются по нескольким приоритетным уровням, в соответствии с требованиями по времени реакции и временем использования процессора.

**Для реализации** приоритетной схемы управления используется ***общий диспетчер прерываний ОС***.

**Замечание** Более подробное рассмотрение алгоритмов взаимодействия подсистемы ввода-вывода с процессором, шиной и контроллерами внешних устройств требует знания и исполь- зования *диаграмм сигналов*, а также знания и практику работы с языком ассемблера. Все эти вопросы выходят за рамки нашего курса.

## BIOS и его функции

**BIOS** (B*asic Input/Output System* ***-*** базовая система ввода-вывода) - часть системного программного обеспечения ЭВМ, реализованная в виде микропрограмм, и обеспе- чивающая для ОС доступ к материнской плате компьютера.

**В персональных** *IBM PC-совместимых компьютерах*, использующих процессоры

*x86*, BIOS записана в микросхему *EEPROM (ПЗУ)* и обеспечивает:

* *начальное тестирование* компьютера;
* *последующую загрузку* ОС.

**Имеется два типа** перезаписываемых микросхем, хранящих BIOS:

* *микросхемы EPROM (Erasable Programmable Read Only Memory)*: содержи- мое этих микросхем может стерто при помощи ультрафиолетового излучения специальным прибором (*старый вариант*);
* *микросхемы EEPROM (Electrically EPROM)*: содержимое этих микросхем мо- жет быть стерто при помощи электрического сигнала, при этом микросхему не обязательно вынимать из компьютера.

Когда появились первые персональные компьютеры, необходимость в BIOS стала критической. Производители компьютеров стали использовать продукты BIOS трех производителей: ***AMI****,* ***AWARD*** *и* ***Phoenix***. Пользователю предоставля- ется **«***Меню***»**, которое позволяет выполнить некоторые специальные настройки компьютера. Для обеспечения выполнения настроек все указанные фирмы исполь- зуют текстовый режим монитора, который именуется *псевдографикой*. Поскольку основные настройки BIOS выполняются самими производителями компьютера, то обычному конечному пользователю следует использовать только две возможности:

* *установка приоритетов* загрузочных устройств;
* *установка адресов* дополнительных плат расширения компьютера.

**Замечание**

Как правило, другие настройки изменять *НЕ РЕКОМЕДУЕТСЯ*!

**Для сохранения** *настроек BIOS* используется микросхема CMOS-памяти.

**CMOS** - ***complementary metal-oxide-semiconductor*** - технология построения элект- ронных схем или КМОП - *комплементарный металлооксидный полупроводник*. **Кроме настроек BIOS** в CMOS хранятся *параметры конфигурации компьютера*.

Суммарный объем памяти CMOS составляет ***256 байт*** и потребляет очень мало энергии. Стандартная батарейка, расположенная на материнской плате питает CMOS в течение ***5-6 лет***, после чего необходимо производить ее замену.

**Замечание**

Если срок батарейки, питающей CMOS, подошел к концу, то при включении ЭВМ на экран будет выведено сообщение, например, "*CMOS-checksum error"*. Для возобновле- ния работы компьютера необходимо будет установить новую батарейку взамен вышед- шей из строя.

**В зависимости** от *версии BIOS* и *модели материнской платы*, функции настройки BIOS могут меняться.

**В разных версиях**, одни и те же функции могут иметь разные названия. **Справочную информацию** по настройке можно найти в инструкции к материнс- кой плате или в сети.

**Замечание**

Программа настройки ***BIOS*** (*BIOS Setup*) может быть вызвана после перезагрузки компьютера нажатием определенной клавиши или группы клавиш.

Наиболее распространенные — **Del** , **F2** или **Esc** .

Существуют также определенные комбинации клавиш, позволяющие:

* *запустить микропрограмму восстановления* (перезаписи) BIOS в микросхе- ме в случае повреждения ее, аппаратно либо вирусом;
* *восстановить заводские настройки*, позволяющие запустить компьютер пос- ле неверных настроек.

### Замечание

*Неверные настройки BIOS могут нарушить работу компьютера*.

## Этапы и режимы POST

Основную часть BIOS материнской платы составляют микропрограммы ини- циализации контроллеров на материнской плате. Подключенные к материнской плате устройства, в свою очередь, могут иметь *управляющие контроллеры с соб- ственными BIOS*.

**Сразу после включения питания компьютера**, во время начальной загрузки ком- пьютера, при помощи программ записанных в BIOS, происходит самопроверка ап- паратного обеспечения компьютера — POST.

**POST** (*Power-On Self-Test*) — самостоятельное тестирование после включения.

Может использоваться *полный* или *сокращенный* тест.

***Сокращенный тест***, включает четыре этапа:

1. Проверку целостности программ BIOS в ПЗУ, используя контрольную сумму.
2. Обнаружение и инициализацию основных контроллеров, системных шин и подключенных устройств: графического адаптера, контроллеров дисководов и другие.
3. Выполнение программ BIOS, обеспечивающих самостоятельную инициали- зацию внешних устройств.
4. Определение размера оперативной памяти и тестирования первого ее сегмен- та: *64 Кбайт*.

***Полный регламент*** работы POST:

1. Проверка регистров процессора;
2. Проверка контрольной суммы ПЗУ;
3. Проверка системного таймера и порта звуковой сигнализации;
4. Тест контроллера прямого доступа к памяти;
5. Тест регенератора генератора оперативной памяти;
6. Тест нижней области ОЗУ для проецирования резидентных программ в BIOS;
7. Загрузка резидентных программ;
8. Тест стандартного графического адаптера (VGA);
9. Тест оперативной памяти;
10. Тест основных устройств ввода (НЕ манипуляторов);
11. Тест CMOS - *Complementary Metal-Oxide-Semiconductor*;
12. Тест основных портов LPT/COM;
13. Тест накопителей на гибких магнитных дисках (НГМД);
14. Тест накопителей на жестких магнитных дисках (НЖМД);
15. Самодиагностика функциональных подсистем BIOS;
16. Передача управления загрузчику ОС.

**Замечание** Выбор между прохождением *полного* или *сокращенного* набора тестов, при включении компьютера, можно задать в программе настройки базовой системы ввода-вывода, *Setup BIOS*.

## UEFI и его стандартизация

Для новых современных платформ ЭВМ, компания *Intel* предлагает **EFI —**

*Extensible Firmware Interface*.

Первоначально, ***середины 1990 годов***, EFI разрабатывалась для первых систем *Intel-HP Itanium*.

Позже, этот интерфейс был переименован в ***UEFI***, разработку которого про- должил *Unified EFI Forum*.

На данный момент, *последняя версия UEFI 2.4*, принята ***в июле 2013 года***.

**Обычно**, UEFI имеет новый графический интерфейс, предполагающий *улучшить*

*«реликтовый BIOS»*. Это стало возможным благодаря совершенствованию техноло- гии изготовления микросхем EEPROM: увеличению их объема и быстродействия, а также снижению себестоимости. Тем не менее, между функционированием BIOS и UEFI имеются существенные различия, которые необходимо хорошо знать. Рас- смотрим это подробнее.

Процессор ***х86***, после включения питания ЭВМ, проводит самотестирование и начинает свою работу *в реальном режиме*, который обеспечивает ему доступ ко всем ресурсам компьютера. Обнулив все регистры, он высталяет значения ***CS*** и ***IP*** специальным образом:

* *для моделей до 80386-DX*: CS=0xFFFF, IP=0x0000 — что указывает на пос- ледние 16 байт в конце 1-го Мбайта оперативной памяти ЭВМ;
* *начиная с 80386-DX*: CS=0x0000, EIP=0xFFFFFFF0 — что указывает на пос- ледние 16 байт в конце 4-х Гбайт оперативной памяти ЭВМ;

**После** установки начальных значений регистров и захвата шины компьютера, процессор начивает выполнять команды извлекаемые из памяти ЭВМ и эта работа не прекращается до полной остановки самого процессора. Обычно, указанные 16 байт, содержат команду ***GOTO*** по адресу ПО BIOS, что поддерживается специ- альной микросхемой памяти, определенной аппаратным конструктивом компью- тера. Таким образом, начинается работа любой современной ЭВМ.

**ПО BIOS**, начиная свою работу, делает небольшой тайм-аут выводит на экран под- сказку, чтобы пользователь мог войти в режим настройки (***BIOS Setup***). Выполнив все программы POST, BIOS ищет загрузочное устройство ЭВМ, среди списка дос- тупных, после чего запускает загрузочный код, расположенный в специальном сек- торе блочного устройства: MBR.

**MBR** — *Master Boot Record* — специальная структура загрузочного устройства, подробно рассмотренная далее.

Возможности ПО BIOS достаточно широки и не ограничиваются только пе- речисленными выше функциями POST, поиском загрузочного устройства и запус- ком программного кода MBR. Чтобы это показать, рассмотрим структуру ОЗУ ЭВМ. Типичная схема оперативной памяти (ОЗУ) IBM PC-совместимого компью- тера показана на рисунке 1.5. В начале ее расположена *область векоторов преры-*

*ваний*, занимающая 1024 байта: по 4 байта на один вектор (всего 256 векторов).

**Вектор прерывания** — адрес программы в памяти ОЗУ (*обработчика прерыва- ния*), которая будет исполняться процессором, когда такое прерывание произойдет.

BIOS в начале работы, выставляет адреса этих векторов на свое собственное ПО, обеспечивая возможности полнофункционального управления компьютером.

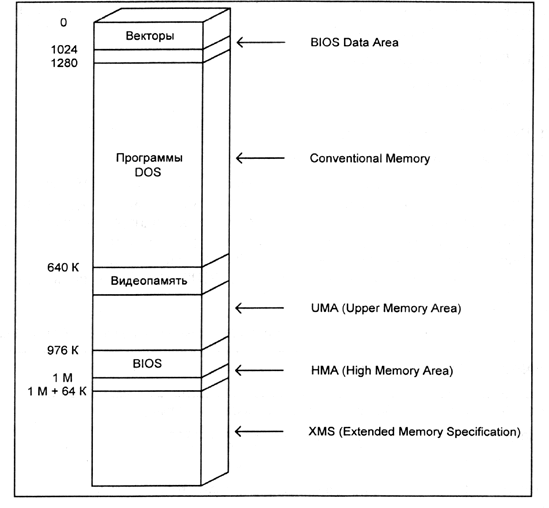


Рисунок 1.5 — Типичная схема ОЗУ памяти ЭВМ

*Для компьютеров IBM PC AT*, назначение ряда векторов прерываний следующее:

* **INT 00h** — деление на 0;
* **INT 01h**— пошаговый режим;
* **INT 02h**— немаскируемое прерывание;
* **ТХТТ ATI INT ОЗh** — точка останова;
* **INT 04h** — переполнение;
* **INT 08h** — таймер;
* **INT 09h** — клавиатура;
* **INT 10h** — видео сервис;
* **INT 33h** — поддержка мыши;
* **INT 4Ah** — будильник пользователя.

Первоначально, такие ОС, как MS DOS, использовали эти прерывания в сво- ей работе. MSDOS имеет даже свой собственный вектор ***21h*** — функции DOS.

**ПО UEFI** интенсивно использует новейшие технологические возможности совре- менных компьютеров. Хотя также выполняются функции аналогичные POST, UEFI сразу переводит процессор в защищенный режим работы, тем самым, обеспечивая повышенную надежность работы ПО. Такой подход позволяет использовать серти- фицированные подходы к использованию ЭВМ, буквально на этапе его включения.

Кроме того, ПО UEFI способно работать с файловыми системами и более современной структурой блочных устройств — GPT.

**GPT** — *GUID Partition Table* — новая структура блочных устройств, позволяющая разбивать их на *128 основных разделов*, в отличие от структуры MBR, которая до- пускает наличие *только 4-х основных разделов*.

**Можно выделить** следующие основные особенности ПО UEFI:

* *работа в защищенном режиме* процессора;
* *возможность защищенной сертификатами* загрузки ОС;
* *возможность поддержки CSM* - *Compatibility Support Module* — модули, обеспечивающие загрузку ОС, совместимую с возможностями BIOS;
* *модульная организация ПО* поддержки аппаратных средств компьютера, *firm- ware*, которое можно собирать и устанавливать от производителей оборудо- вания;
* *специальная структура ПО*, соответствующая запускаемым *exe-файлам* ОС MS Windows;
* *возможность загрузки множества* разных ОС;
* *непосредственный вызов* загрузчика ОС;
* *возможность работы* со структурой GPT блочных устройств;
* *требование наличия собственного специального раздела* для блочных уст- ройств;
* *требование формата FAT12/16/32* для собственного раздела.

### Замечание

1. Стали появляться ноутбуки с UEFI и предустановленной ОС MS Windows 8. При покупке такого компьютера, убедитесь, что основное окно UEFI имеет переключа- тель «*Legacy BIOS*» и отключение режима «*Security*», иначе вам не удастся загру- зить другую ОС.
2. Для компьютеров на базе иных платформ, чем ***х86***, для обозначения встроенного ПО могут использоваться другие термины. Например, для архитектур *SPARC* или *UltraSPARC*, *firmware* может называться ***PROM*** или ***Boot***.

## Блочные и символьные устройства компьютера

В предыдущей теме, перечисляя типы специальных файлов, мы отметили

*файлы устройств*.

Все файлы устройств разделяются на ***символьные*** и ***блочные***:

* *блочными* называются внешние устройства компьютера, обмен данными с которыми можно производить только блоками: *целостной упорядоченной последовательностью байт*; к блочным устройствам относятся «жесткие» и

***floppy***-диски, магнитные ленты, диски CDROM и другие; на блочных уст- ройствах возможно создание *файловых систем*;

* *все другие устройства*, не являющиеся блочными, называются *символьными*; обмен данными с символьными устройствами осуществляется по *одному*

*байту*; например, клавиатура, мышь, консоль экрана, COM-порты, сетевые устройства и другие — *символьные*.

**Замечание**

Магнитные ленты могут иметь *физические блоки переменной длины*.

«Жесткие диски (вичестера)» имеют *физические блоки фиксированной длины*. Текущий стандарт физического блока винчестера: *1 сектор — 512 байт*.

Символьное устройство *не обозначает текстового содержимого*.

Для символьных устройств, во многих случаях, понятие объема хранения данных не применимо.

***ОС MS Windows*** обозначает разделы блочных устройств, имеющих форматы FAT и NTFS буквами с двоеточием: *A:, B:, C:, …, Z:*. Прописные и заглавные буквы

* неразличимы. Символьные устройства обычно скрыты за графическим интер- фейсом и, в явном виде, не используются.

***ОС UNIX и Linux*** имеют общие правила обозначения устройств:

* + имена устройств находятся в специальной директории ***/dev***; в нее смонтиро- вана специальная область ядра ***dev*** с файловой системой типа *devtmpfs*;
  + имена устройств имеют имя драйвера, которое управляет этим устройством; прописные и заглавные буквы различаются;
  + имена устройств, объединенных одним драйвером, разделяются цифрой, до- бавляемой к имени драйвера, начиная с нуля.

**Замечание**

При наличии соответствующих драйверов, в ОС UNIX и Linux, можно с блочным устрой- ством работать как с символьным, поэтому понятия блочный и символьный применимы и к драйверам, управляющим устройствами.

## Винчестер и загрузочные устройства

Традиционно, загрузочным устройством ЭВМ является *винчестер* или

*«жесткий диск»*.

Конструктивно, винчестер состоит из набора круглых пластин, которые цент- ральной частью, на некотором расстоянии, надеты на шпиндель, вращающийся посредством электродвигателя:

* + *каждая сторона круглой пластины* покрыта магнитным составом, способ- ным фиксировать информацию посредством *магнитных головок*, которые

«плавают» над каждой стороной диска;

* + *отдельная окружность* на отдельной стороне диска образует *трек (track)*;
  + *совокупность треков одного диаметра*, образуют *цилиндр (cylinder)*; цилинд- ры пронумерованы от внешнего края диска, *начиная с 0*;
  + *все магнитные головки винчестера (head)* одновременно находятся над тре- ками одного цилиндра и пронумерованы от 0 (*обычно от 0 до 15*);
  + *каждый трек разделен на 63 части* — *сегменты (физический блок)*, прону- мерованные, *начиная с 1*.

**Таким образом**, физические блоки винчестера (*сегменты*) пронумерованы в систе- ме координат ***CHS***, начиная с сегмента (*0, 0, 1*).

**Замечание** Следует отметить, что *адресация CHS* (*Цилиндр, Головка, Сектор*), заложенная в конструкцию первых персональных компьютеров и ПО BIOS, не позволяет адресовать *более 7.8 Гбайт* данных. Поэтому современные ЭВМ, имеющие винчестера емкостью более 7.8 Гбайт, используют *адресацию LBA*.

**LBA** (*Logical block addressing*) — механизм адресации и доступа к блоку данных на на «жестком диске», при котором *системному контроллеру* нет необходимости учитывать геометрию самого жесткого диска: количество цилиндров, сторон и секторов на цилиндре. Контроллеры современных IDE-дисков в качестве основного режима трансляции адреса используют LBA.

**Суть LBA** состоит в том, что *каждый блок, адресуемый на жестком диске, имеет свой номер* - целое число, *начиная с нуля* и далее:

### LBA 0 = Цилиндр 0/Головка 0/Сектор 1

*Преимущество метода адресации LBA* — ограничение размера диска обусловлено лишь разрядностью LBA. В настоящее время, для задания номера блока использу- ется *48 бит*, что дает возможность адресовать (248) 281 474 976 710 656 блоков.

*Технический комитет X3T10* установил правила получения адреса блока в режиме LBA, при условии, что размер блока равен размеру сектора:

***LBA***( ***c , h , s***)=( ***c***∗ ***H*** + ***h***)∗ ***S***+ ***s***+ **1**

***s***=( ***LBA mod S***)+ **1**

***h***=( ***LBA***+ **1**− ***s***) ***mod*** ( ***H***∗***S***)

***S***

***c***= ***LBA***+ **1**− ***s***−***h***∗***S***

***H***∗ ***S***

где:

**с** — номер текущего цилиндра;

**h** — номер текущей головки; **s** — номер текущего сектора; ***H*** — число головок;

***S*** — число секторов на дорожке;

***mod*** — операция взятия остатка от деления.

**После того**, как BIOS закончит начальный тест POST, она начнет просматривать блочные устройства ЭВМ с целью поиска *загрузочного устройства* ОС:

* + Блочные устройства просматриваются в том порядке, который указан в наст- ройках BIOS. Чтобы определить является ли устройство загрузочным, BIOS

читает первый сектор блочного устройства и помещает его в ОЗУ ЭВМ. В компьютерах архитектуры IBM PC, этот адрес обычно ***0000****:****7c00***.

* + Если сектор соответствует MBR — *Master Boot Record*, то BOIS передает управление его загрузочному коду: обычно командой *long jump*.
  + Если структура прочитанного сектора не соответствует MBR, то проверяется следующее устройство.
  + Если все просмотренные устройства не являются загрузочными, то BIOS: или перезапускает ЭВМ или загружает встроенный в BIOS интерпретатор языка BASIC (если он, конечно, — есть).

### Замечание

*BIOS* рассматриватет flashUSB как загрузочное устройство, если:

* + его ПО поддерживает такие устройства;
  + имеется раздел MBR, который отмечен как загрузочный.

*UEFI* рассматривает flashUSB как загрузочное устройство, если оно имеет раздел,

форматированный как FAT12/16/32, и в корне разделя имеется директория *EFI*.

## Загрузочный сектор MBR, его назначение и архитектура

***MBR* (M*aster Boot Record)* —** это *Главная загрузочная запись* блочного устройства. *Более точно* — это код и данные расположенные в первом секторе блочного устройства, которые могут быть использованы для загрузки некоторой ОС.

**В общем случае**, под *загрузчик MBR* выделено ***32 Кбайт*** винчестера или другого внешнего блочного накопителя. Если под загрузчик ОС используются все 32 Кбайт, то под MBR понимают весь этот загрузочный код. В этом случае, ***первые 512 байт*** называют *MBS — Master Boot Sector* или *главным загрузочным сектором*. Для опе- рационных систем *MS Windows*, понятия MBR и MBS совпадают, так как вся MBR содержится в MBS и они рассматриваются как синонимы.

**Замечание** MBR может не содержать загрузочного кода, если блочное устройство не является загру- зочным. Более того, сам термин появился в те времена, когда:

* + с одного устройства загружалась только одна ОС;
  + структура блочного устройства была уникальна для каждой ОС.

Последующая унификация структур блочных устройств и самих загрузочных записей

привели к тому, что ***MBR*** — это еще не загрузка ОС, а всего лишь выбор: *«с какого раздела жесткого диска следует загружать ОС»:*

* + *На стадии MBR* происходит только выбор раздела диска и ничего более.
  + *Загрузка самой ОС* происходит на более поздних этапах.

*Структура MBR* содержит три основные части (см. таблицу 1.1):

* + *небольшой фрагмент* исполняемого кода, - 446 байт;
  + *таблицу* разделов (*partition table*);
  + *специальную сигнатуру*.

|  |  |
| --- | --- |
| *Таблица 1.1 - Структура Главной загрузочной записи (MBR)* | |
| ***Адрес*** | ***Содержимое*** |
| 0x0000 | Код загрузчика (446 байт) |
| 0x01B8 | 4-х байтная сигнатура диска (только для MS Windows 2000 и XP) |
| 0x01BE | Четыре 16-байтных записи схемы таблицы основных разделов MBR  **(partition table)** |
| 0x01FE | 2-х байтная сигнатура MBR (55AA16) |

Поскольку утвержденного стандарта на структуру MBR не существует, то используется «*стандарт де-факто*», распространенный Microsoft, и которого при- держиваются большинство дистрибьютеров ОС.

Согласно «*традиции MBR*», винчестер может быть *разбит на четыре основных раздела*. Допускается один из разделов использовать как *расширенный раздел* и делить его дополнительно. Традиционно также, *MBR создается или редактируется в момент инсталляции ОС на внешний носитель*.

**Когда BIOS** прочитает первый сектор блочного устройства и запишет его по адресу

***0000:7С00***, она проверяет наличие сигнатуры ***55AAh***:

* + Если сигнатура ***есть***, управление передается коду загрузчика MBR;
  + Если сигнатуры **нет**, то проверяется следующее блочное устройство.

*Код загрузчика MBR*:

* + *копирует себя* с адреса ***0000:7C00*** по адресу ***0000:6000***, освобождая место для непосредственного загрузчика ОС;
  + *работает с таблицей разделов* (partition table), структура отдельной строки которой показана в таблице 1.2: *если загрузочный раздел найден*, то первый сектор загрузчика ОС записывается по адресу ***0000:7С00*** и ему передается

управление; *если загрузочный раздел не найден* или обнаружена ошибка за- писи *partition table*, то делается прерывание **INT 18h** и управление передает- ся назад в BIOS.

*Для отдельной записи partition table*:

*Первый байт* содержит ***признак активности раздела***: признак, обозначающий возможность загрузки операционной системы с данного раздела. Для стандартных загрузчиков может принимать следующие значения:

* + **80h** — раздел является активным;
  + **00h** — раздел является неактивным;
  + ***Другие*** значения являются ошибочными и игнорируются.

*Следующие три байта* задают начало раздела в системе координат (***C,H,S***).

*Пятый байт* обозначает *код файловой системы:* Partition Ids, некоторые значения которого приведены в таблице 1.3.

*Следующие три байта* задают окончание раздела в системе координат (***C,H,S***). *Завершают строку partition table* два четырехбайтовых числа, задающие начало раздела и его длину в секторах.

|  |  |  |
| --- | --- | --- |
| *Таблица 1.2 - Структура описания раздела* | | |
| ***Смещение*** | ***Длина*** | ***Описание*** |
| 00h | 1 | ***Признак активности*** раздела |
| 01h | 1 | Начало раздела — головка |
| 02h | 1 | Начало раздела — сектор (биты 0-5), дорожка (биты 6,7) |
| 03h | 1 | Начало раздела — дорожка (старшие биты 8,9 хранятся в байте номера сектора) |
| 04h | 1 | ***Код типа раздела*** – код файловой системы |
| 05h | 1 | Конец раздела — головка |
| 06h | 1 | Конец раздела — сектор (биты 0-5), дорожка (биты 6,7) |
| 07h | 1 | Конец раздела — дорожка (старшие биты 8,9 хранятся в байте номера сектора) |
| 08h | 4 | Смещение первого сектора раздела блочного устройства |
| 0Ch | 4 | Количество секторов раздела |

*Таблица 1.3 - Ранее распространенные коды типов файловых систем*

|  |  |
| --- | --- |
| ***ID (hex)*** | ***Описание*** |
| 01 | Primary DOS12 (12-bit FAT) |
| 04 | Primary DOS16 (16-bit FAT) |
| 05 | Extended DOS |
| 06 | Primary big DOS (> 32MB) |
| 0A | OS/2® |
| 83 | Linux (EXT2FS) |
| A5 | FreeBSD, NetBSD, 386BSD (UFS) |

**Замечание**

Допускается, чтобы один из разделов блочного устройства имел код типа файловой системы равный ***05h***, который соответствует структуре раздела ***EBR — Extended Boot Record***, начинающегося со структуры, приведенной в таблице 1.4.

|  |  |  |
| --- | --- | --- |
| *Таблица 1.4 - Структура EBR* | | |
| ***Смещение*** | ***Длина*** | ***Описание*** |
| 1BEh | 16 | Указатель на раздел |
| 1CEh | 16 | Указатель на следующий EBR |
| 1FEh | 2 | Сигнатура (55h AAh) |

**Замечание**

Формат указателей в таблице 1.4 аналогичен формату строки *Partition Table* в MBR. Раздел EBR не может содержать в себе других разделов EBR.

Традиционная таблица разделов винчестера MBR ориентирована на загрузку только одной ОС.

Дальнейшее развитие структуры блочных устройств связано с созданием новой усовершенствованной таблицы разделов: **GPT**, показанной на рисунке 1.6.

**GPT** — ***GUID Partition Table***.

**GUID — Global Unique Identifier** - глобальный уникальный идентификатор, кото- рый в данном контексте используется для именования разделов блочных устройств:

* + *само устройство* именуется в момент создания на нем струтуры GPT;
  + *раздел устройства* именуется в момент создания на нем файловой системы.

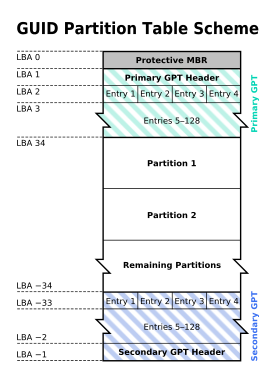


Рисунок 1.5 - Структура таблицы разделов GPT

*Теперь*,

* + каждое блочное устройство может быть разбито на ***128 разделов***, записи о которых дублируются в конце устройства **(***отрицательные номера LBA***)**.
  + Все разделы GPT являются основными.
  + Размер ***LBA=512 байт*** и ***LBA0*** — для совместимости соответствует *MBR*.
  + Отдельный LBA содержит *записи о 4-х разделах*: по 128 байт на раздел.
  + Для идентификации раздела используются GUID.

**Замечание** *MBR*, входящий в структуру GPT, должен указывать наличие на блочном устройстве только одного раздела с типом ***eeh*** и размером:

* + *всего блочного устройства*, для устройств < 2 ТБайт;
  + *2 ТБайт*, для устройств > 2 ТБайт.

*Не все BIOS* могут работать со структурой GPT.

## GRUB как универсальный загрузчик ОС

**Проблема загрузки разных ОС** с одного блочного устройства всегда будоражила умы пользователей ЭВМ, тем более, что емкость винчестеров постоянно увеличи- валась. Хотя появились UEFI и структура блочных устройств GPT, имеется ряд проблем, в том числе и организационных, которые делают необходимым примене- ние различных универсальных загрузчиков.

*Наиболее распространенными* загрузчиками ОС являются:

* + NTLDR— загрузчик ядра MS Windows NT;
  + Windows Boot Manager (bootmgr.exe, winload.exe) — загрузчик ядра MS Windows Vista; *bootmgfw.efi — для MS Windows 7, 8, 8.1 и других*;
  + LILO (LInux LOader) — старый загрузчик ядра Linux;
  + *GRUB* (Grand Unified Bootloader) — новый загрузчик ядра Linux и Hurd;
  + RedBoot — загрузчик для встраиваемых систем;
  + SILO (SPARC Improved bootLOader) — загрузчик Linux и Solaris для машин с архитектурой SPARC;
  + Loadlin — загружает Linux из под MS DOS или MS Windows;
  + Syslinux — загружает Linux из под MS DOS или MS Windows;
  + BOOTP — применяется для загрузки по сети.

*Среди перечисленного ПО*, наиболее интересным является ***GRUB*** — официальный загрузчик Linux из проекта GNU. Он может загружать разные ОС, включая MS Windows, с многих разных аппаратных платформ. GRUB (точнее ***GRUB2***) входит и в дистрибутив ***Arch Linux***. Он устанавливается на ЭВМ в процессе инсталляции ОС. Это позволяет свободно использовать и Linux и MS Windows на одном компьютере.

*Более подробно*, вопросам работы с ПО GRUB посвящена лабораторная работа №2, которая входит в программу обучения по данной теме. В качестве дополнительного учебного материала, которым необходимо пользоваться как справочной информа- цией, является пособие [4, раздел 2].

## Меню и функции GRUB

**ПО GRUB** можно рассматривать как маленькую однопользовательскую ОС специ- ального назначения: *интерактивная загрузка различных ОС*.

**Многие идеи** этого ПО и, в частности модульная организация, использовались разработчиками ***UEFI***.

*Среди основных функций GRUB* следует выделить:

* + *поддержка интерпретатора сценариев*, близких по функциональным воз- можностям языка ***shell***, который собственно говоря и был его прототипом;
  + *умение работать со структурами* ***MBR*** и ***GPT*** блочных устройств;
  + *поддержка работы* со многими современными устройствами ЭВМ;
  + *распознавание и умение работать* с многими современными файловыми сис- темами;
  + *поддержка национальных языков* и других мультимедийных средств ЭВМ;
  + *обнаружение на блочных устройствах ЭВМ* наличия различных ОС и авто- матическое формирование для них сценария меню загрузки.

*Хотя не все функции* GRUB работают одинаково эффективно, потенциал этого ПО является очевидным.

*Основная часть* ПО GRUB располагается в разделе блочного устройства и устанв- ливается в процессе инсталляции на него ОС Linux. По умолчанию, оно помещает- ся в директорию */boot/grub*, туда же помещается автоматически созданный файл конфигурации *grub.cfg*.

**Дополнительно**, в процессе инсталляции ОС анализируется наличие ***UEFI*** и выполняется необходимая работа с ним.

*Важно помнить*, что дистрибутивы ***GRUB*** различаются для разных архитектур процессора ЭВМ, хотя допускается их смешанная установка на один компьютер.

**Также**, имеются варианты дистрибутивов для работы с ЭВМ, имеющими ***UEFI***.

**Замечание** При повторной инсталляции разных дистрибутивов ОС будет установлена и соответ- ствующая версия ПО GRUB и частично могут быть изменены уже имющиеся его наст- ройки. В первую очередь, это касается файла *grub.cfg*, поэтому следует заранее позабо- титься о его сохранности.

Если на ЭВМ установлено несколько дистрибутивов ПО GRUB, то в процессе загрузки будет работать только один из них, *хотя файл grub.cfg* ***может быть общий!***

*Одним из вариантов* использования ПО GRUB является установка его на ***flashUSB***, что и применяется в процессе обучения по нашей дисциплине.

**Такой подход** позволяет избежать многих проблем, связанным с недостаточной квалификацией исполнителей.