ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО СВЯЗИ
СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ

С.Н. Мамойленко

ОРГАНИЗАЦИЯ ЭВМ и СИСТЕМ
Практикум

Новосибирск
2005

УДК 681.3.068(075)

Ктн С.Н. Мамойленко. Организация ЭВМ и систем: Практикум/СибГУТИ.- Новосибирск, 2005г.- 86 стр.

Практикум предназначен для студентов, обучающихся по направлению 230100 «Информатика и вычислительная техника» и изучающих дисциплину «Организация ЭВМ и систем». В нём содержится материал, предназначенный для проведения практических или лабораторных занятий по указанному учебному курсу с целью изучения основных принципов построения персональных компьютеров и некоторых периферийных устройств.

Кафедра вычислительных систем
Ил. 43, табл. 3, список лит. – 8 назв.

Рецензент: О.А. Бах

Для специальностей 230101 – «Вычислительные машины, комплексы, системы и сети» и 230105 – «Программное обеспечение вычислительной техники и автоматизированных систем».

Утверждено редакционно-издательским советом СибГУТИ в качестве учебного пособия.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Электронно-вычислительная машина (ЭВМ) или компьютер (англ. computer – вычислитель) – это аппаратурно-программный комплекс, предназначенный для обработки информации. Под обработкой понимается: преобразование (т.е. выполнение некоторых вычислительных операций), а также ввод, вывод и хранение информации.

Подобные устройства нашли применение практически во всех областях деятельности человечества. Изначально использовались большие сложные вычислительные машины, затем более широкое распространение получили персональные компьютеры.

Персональный компьютер (ПК) (англ. personal computer, PC) – это массово применяемая ЭВМ, имеющая небольшие габаритные размеры и невысокую стоимость.

В рамках подготовки специалистов по направлению 230100 «Информатика и вычислительная техника» изучению принципов построения вычислительных машин и систем уделяется особое внимание. Согласно государственному образовательному стандарту первое знакомство с этими принципами студенты получают в рамках курса «Организация ЭВМ и систем», отнесённого к блоку общепрофессиональных дисциплин. Очевидно, что для студентов, ограничиваться только получением теоретических навыков нецелесообразно. Важным является организация практических занятий помогающих студентам закрепить полученные теоретические знания.

В указанном направления имеется несколько специальностей, в том числе 230101 – «Вычислительные машины, комплексы, системы и сети» и 230105 – «Программное обеспечение вычислительной техники и автоматизированных систем». Именно для студентов, обучающихся по этим специальностям, предназначено это учебное пособие.

В учебном пособии представлен материал, предназначенный для организации практических занятий по изучению основных принципов работы персональных компьютеров, а также получению базовых навыков системного программирования в UNIX-подобных операционных системах.

В приложении приведены задания на лабораторные работы, которые в рамках курса «Организация ЭВМ и систем» выполняют студенты, обучающиеся на кафедре вычислительных систем Сибирского государственного университета телекоммуникаций и информатики.

ГЛАВА 1. КРАТКАЯ ИСТОРИЯ ВОЗНИКНОВЕНИЯ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Первым шагом, сделанным на пути создания ПК, считается ЭВМ Altair-8800, созданная в 1975 компанией Micro Instrumentation and Telemetry Systems (MITS), расположенной в Соединенных штатах Америки (в городе Альбукерке, штат Нью-Мексико) (см. рис. 1).

Рис. 1. Микрокомпьютер Altair-8800

ЭВМ Altair-8800 (тогда она называлась микрокомпьютером) была основана на микропроцессоре 8080 фирмы Intel, оснащена блоком питания, лицевой панелью с множеством индикаторов и оперативной памятью емкостью 256 байт (!). Весь комплект тогда стоил 397 долларов и продавался в виде набора деталей, которые покупатель должен был самостоятельно собрать, используя паяльник. Программы в ЭВМ вводились переключением тумблеров на передней панели, а результаты считывались со светодиодных индикаторов.

Микрокомпьютер Altair 8800 был построен по принципу открытой архитектуры с использование общей шины, что позволило пользователям самостоятельно разрабатывать дополнительные платы для подключения внешних устройств.

Название "Altair" придумала дочь Эда Робертса, возглавлявшего в то время компанию MITS. Так называлась звезда из популярного тогда телесериала "Звёздный поход".

В 1976 году компания - Apple Computer, выпустила свою модель персонального компьютера - Apple I (см. рис. 2), стоимостью 695 долларов. Его системная плата была прикручена к куску фанеры и полностью отсутствовали корпус и блок питания.

Настоящий успех к компании Apple Computers пришел с выводом на рынок модели компьютера Apple II (см. рис. 3). Это был первый в истории персональный компьютер в пластиковом корпусе и с цветным экраном. Стоил он тогда больше тысячи долларов. В начале 1978 года на рынок вышел недорогой дисковод для дискет Apple Disk II, который еще больше увеличил популярность этого ПК.

Рис. 2. Компьютер Apple I

Рис. 3. Компьютер Apple II

Рис. 4. Филипп Дон Эстридж. Руководитель подразделения IBM, в котором был создан первый IBM PC

В начале 80-х годов XX столетия компания International Business Machines (IBM) основала в штате Флорида в городе Бока-Ратон (Boca-Raton) отделение Entry System Division, объединившее группу из 12 человек под руководством Филиппа Дона Эстриджа (Phillip Don Estridge). Главным конструктором был назначен Левис Эггебрехт (Lewis Eggebrecht).

Целью создания этой лаборатории была разработка ЭВМ, доступной широкому кругу потребителей, которая должна иметь небольшие размеры, невысокую стоимость и производительность, позволяющую решать определённый набор задач. Другими словами, целью создания это лаборатории была разработка персонального компьютера.

Первый IBM PC (см. Рис. 5) появился 12 августа 1981 года и стал родоначальником нового стандарта построения ЭВМ.

Рис. 5. Первый IBM PC

Рис. 6. Современный персональный компьютер семейства IBM PC

С тех пор уже прошло более двадцати лет, и, конечно же, многое изменилось (см., например, внешний вид современного ПК, Рис. 6). Например, производительность (число операций, выполняемых в единицу времени) современных ПК по сравнению с первыми возросла в тысячи и более раз.

На сегодняшний день фирма IBM не является единственным производителем персональных компьютеров. Однако, используя термин *персональный компьютер,* часто имеют в виду именно IBM-совместимый ПК.

ГЛАВА 2. ОБЩАЯ ОРГАНИЗАЦИЯ СОВРЕМЕННЫХ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Современный персональный компьютер с архитектурой IBM PC (далее для краткости будем его просто называть персональный компьютер или ПК) в общем случае состоит из системного блока и внешних устройств (монитора, клавиатуры, манипулятора «мышь», колонок и т.п.). По сути, компьютером является то, что содержится в системном блоке, а именно:

- процессор;
- оперативная память;
- набор микросхем системной логики,
- слоты расширения.

Кроме этого системный блок содержит корпус, блок питания, периферийные устройства (жесткий диск, дисковод гибких дисков, CD-ROM и т.п.).

Все устройства, входящие в состав ПК, создаются на основе стандартов, называемых **форм-факторами** (англ. form-factor). Они определяют геометрические размеры изделия и интерфейс взаимодействия с другими узлами ПК. Форм-факторы для узлов ПК будут рассмотрены ниже.

2.1. Корпус персонального компьютера

Корпус (см. рис. 7) - это кожух, внутри которого размещаются все основные компоненты персонального компьютера. Форм-факторы корпусов задают их геометрические размеры, форму и способы для крепления всего, что будет составлять системный блок, виды и расположение интерфейсных выводов, тип блока питания и способ его включения (тумблером или сигналом от материнской платы).

Корпусы бывают настольного (напольного) и стоечного исполнения. Первые предназначены для использования на рабочих местах, вторые для установки в специальные стойки, объединяющих несколько ПК и другое оборудование.

Корпусы с настольным исполнением могут располагаться горизонтально (типа «рабочий стол», англ. desktop) или вертикально (типа «башня», англ. tower).

По высоте «башенные» корпусы бывают: мини- (англ. mini), микро- (англ. micro), миди- (англ. midi), большие- (англ. big). Высота определяется числом отсеков, предназначенных для установки периферийных устройств с форм-факторов 5,25 дюйма (дисководы магнитных дисков, CD-ROM и т.д.). Кроме этого, существуют тонкие (slim) корпусы башенного типа. В таких корпусах отсеки 5,25 располагаются вертикально, а ширина корпуса определяется устройствами с форм-факторов 3,5 дюйма (дисководы, ZIP-диски и т.п.).

Горизонтальные корпусы различаются по видам системных плат, которые можно в них установить. Бывают: маленькие- (англ. mini), большие- (англ. full).

Кроме рассмотренных (классических) корпусов в последнее время на рынке появились различные их модификации, например, в виде музыкальных центров, автомобильных приборных панелей и т.п.

rack-mount

Slim

Рис. 7. Стандарты для корпусов персональных компьютеров

Слева корпусы башенного типа (слева направо - мини-, микро-, миди-, большие-, рабочий стол (сверху мини, снизу большой). Справа вверху корпус для установки в специальную стойку. Справа внизу – корпус типа slim-башня**.**

2.2. Системная (материнская) плата персонального компьютера

Системная плата (см. рис. 9) представляет собой многослойную плату, объединяющую все электрические схемы, которые, в совокупности, и образуют вычислительную машину. Кроме электрических схем на ней располагаются разъемы для подключения процессора, памяти и периферийных устройств, а также некоторые переключатели. Структуру системной платы определяют состав смонтированных на неё электронных компонентов – **чипсет (**англ. chipset) и, конечно же, форм-фактор.

На сегодняшний день существуют следующие форм-факторы материнских плат: AT (Baby-AT), ATX (miniATX, microATX, FlexATX), LPX, NLX (microNLX), WPX. Конечно же это далеко не полный их перечень.

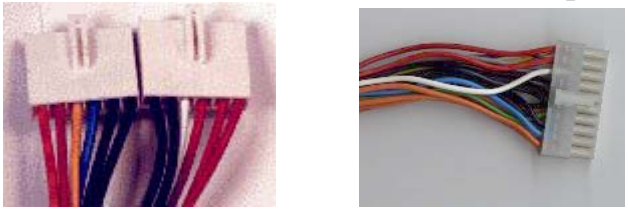


Рис. 8. Разъемы для подключения блоков питания к материнским платам (слева – AT, справа ATX)

Форм-фактор AT делится на две отличающиеся по размеру модификации - AT и BabyAT. Размер полной AT платы до 12" в ширину и до 13” в длину. BabyAT имеет размер 8.5" в ширину и 13" в длину. У всех плат стандарта AT последовательные и параллельные порты присоединяются к материнской плате через специальные планки, имеется один разъем клавиатуры типа DIN (круглый 5-ти штырьковый). Для подключения блока питания используется 12-

штырьковый разъем (см. рис. 8), состоящий из двух частей (по 6 линий каждый). Устанавливать его надо так, чтобы все черные провода находились в центре. Управление блоком питания не поддерживается.

Форм-фактор ATX предложен компанией Intel в 1995 году. В таких платах все порты ввода-вывода (последовательные и параллельные) интегрированы прямо на плату, присутствуют разъемы типа PS/2 (круглый 6-ти штырьковый) для клавиатуры и мышки и USB для подключения внешних устройств (см. рис. 9). Введены дополнительные каналы для взаимодействия с блоком питания, используя для подключения последнего один 20-контактный разъем. Одновременно с ATX появились различные модификации формата: Full-ATX (305x244), Mini-ATX (284x208), Micro-ATX (244x244), Flex-ATX (229x203). Различаются они лишь размерами материнской платы (количеством слотов на ней и более плотной компоновкой всех элементов). Для мощных, больших компьютеров (серверов) создали спецификацию EATX (Extended ATX), но он не получил широкого распространения и был вытеснен форм-фактором WTX (Workstation Technology eXtended).

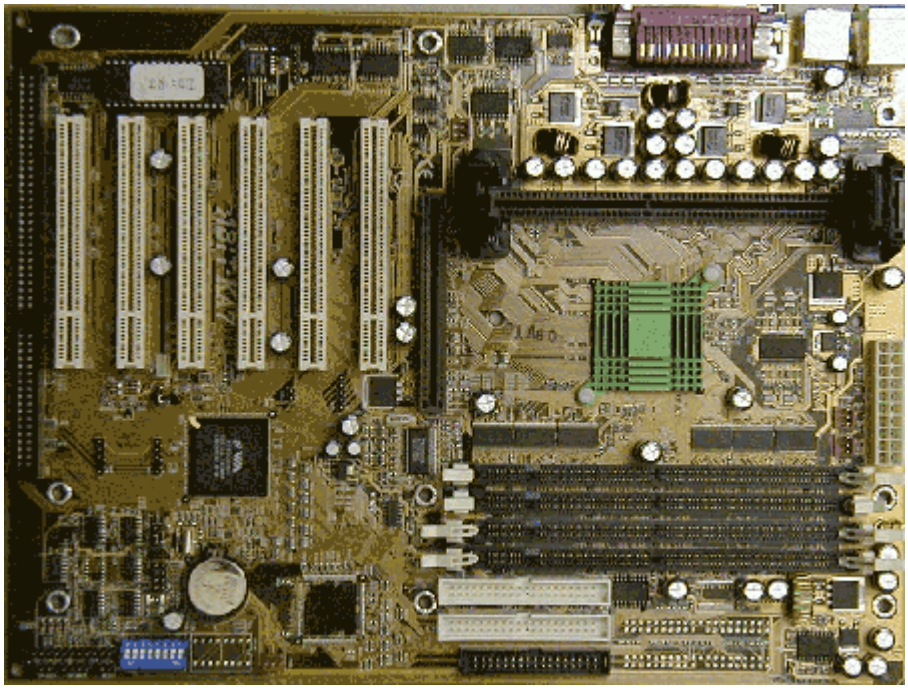


Рис. 9. Пример системной платы (форм фактор FullATX)

В узких (Slim) корпусах стал применяться **форм-фактор LPX**. Сокращение размеров была реализовано путем введения специальной вертикальной стойки, к которой подключаются все внешние устройства, а сама стойка, в свою очередь, подключается к материнской плате. Это позволило заметно уменьшить высоту корпуса, поскольку обычно именно высота карт расширения влияет на этот параметр. Расплатой за компактность стало максимальное количество подключаемых карт - 2-3 штуки. Размер материнской платы 330x229 мм. Позже появился форм-фактор Mini-LPX с размерами материнской платы 264x201 мм.

На смену стандарту LPX пришел **форм-фактор NLX**. Основной идеей при разработке этого стандарта было удобство сборки и обслуживания персо-

нального компьютера. Для этого корпус, системная плата и вертикальная стойка оформляются таким образом, что материнская плата свободно отсоединяется от стойки и выдвигается из корпуса. Фактически стойка - это одна материнская плата, разделенная на две части - часть, где находятся собственно системные компоненты, и подсоединенная к ней через 340-контактный разъем под углом в 90 градусов часть, где находятся всевозможные компоненты ввода/вывода - карты расширения, разъёмы портов, накопителей данных, куда подключается питание. Таким образом, во-первых, повышается удобство обслуживания - нет необходимости получать доступ к ненужным в данный момент компонентам. Во-вторых, производители в результате имеют большую гибкость - делается одна модель основной платы и стойка под каждого конкретного заказчика, с интеграцией на ней необходимых компонентов. Размеры материнской платы стандарта NLX 345x229 мм. Также имеется уменьшенный вариант платы с форм-фактором Mini-NLX размером 254x203 мм.

В 1998 году появился **форм-фактор WTX, о**риентированный на поддержку двухпроцессорных материнских плат любых конфигураций, поддержку огромного количества периферийных устройств. Основной целью стандарта было определение правил организации теплообмена в высокопроизводительных ЭВМ.

2.3. Набор микросхем системной логики (chipset)

Сама по себе материнская плата, с подключенными к имеющимся на ней разъёмам внешним устройствами, ещё не является вычислительной машиной. Для организации взаимодействия между всеми этими узлами используется набор микросхем системной логики, называемый также **чипсетом** (англ. chipset). Для этого в его состав входят контроллеры прерываний, доступа к памяти, управления интерфейсами с внешними устройствами (последовательный и параллельный интерфейс, PCI, AGP, ISA, IDE и т.п.), часы реального времени и т.д. Состав и структура набора микросхем системной логики определяет основные характеристики ПК в целом.

Существует много разновидностей наборов микросхем системной логики. Их состав зависит от фирмы производителя и от того, какое оборудование поддерживается этими наборами.

В общем случае наборы микросхем системной логики строятся по принципу двух мостов – северного и южного (см. рис. 9, 10). Так они названы потому, что на структурной схеме один мост изображается наверху (север), а другой внизу (юг). Использование двух мостов продиктовано соображениями увеличения производительности. К одному мосту подключаются устройства, медленно передающие информацию, к другому – передающие быстро. Между собой мосты взаимодействуют также через шину.

Северный мост (англ. north bridge) предназначен для организации взаимодействия между наиболее быстрыми узлами ПК, такими как процессора, оперативной памяти, ускоренного графического порта AGP, интерфейса с внешними устройствами PCI (Peripheral Component Interconnect, 33 МГц). Чаще всего он работает на полной тактовой частоте системной платы (на частоте ши-

ны процессора). В функции Северного моста входит управление потоками данных из 4-х шин: шина памяти, AGP, системная шина процессора и шина связи с Южным мостом. В современных наборах микросхем северный мост реализован в виде одной микросхемы (на одном кристалле), раньше же требовалось до трех микросхем для его реализации.

Южный мост (англ. south bridge) обслуживает медленные устройства, подключаемые через шину ISA (8МГц), контроллер жесткого диска IDE и USB (Universal Serial Bus), Также он содержит в себе схемы реализующие функции памяти CMOS и часов, контроллер прямого доступа в память и контроллер прерываний.

В дополнение к мостам используется третья микросхема, называемая Super I/O и содержащая в себе контроллеры для управления некоторыми внешними устройствами (дисководом, последовательным и параллельным портами и т.п.), а также базовая система ввода вывода (BIOS), энергонезависимая память, содержащая основные параметры и часы реального времени (CMOS). Базовая система ввода-вывода содержит программу начального тестирования POST (Power-On Self Test), программу начальной загрузки, драйверы для интегрированных устройств.

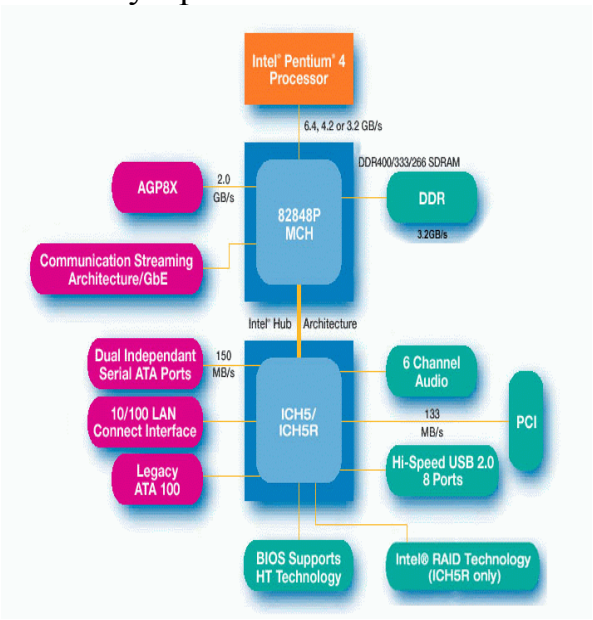


Рис. 10. Набор микросхем системной логики (Intel 845)

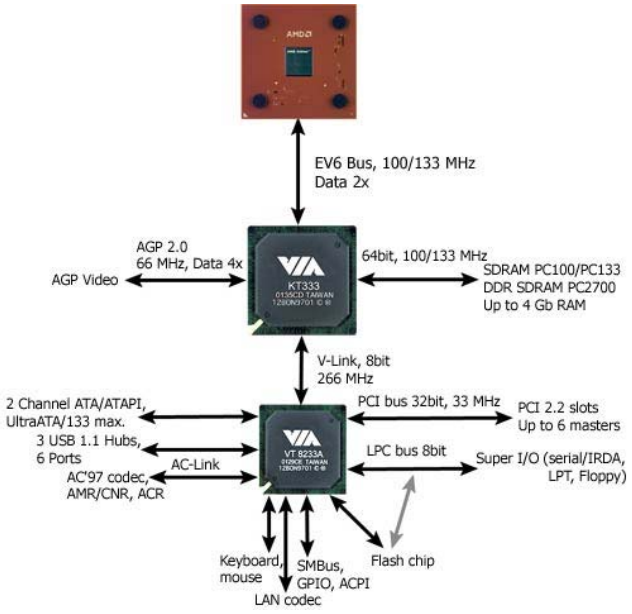


Рис. 11. Набор микросхем системной логики (VIA Apollo Pro)

С недавнего времени (с выпуском корпорацией Intel чипсета i815) корпорация Intel отказалась от использования архитектуры мостов (см. рис. 12), и перешла к похожей архитектуре, в которой используются концентраторы - хабы. На первый взгляд все то же самое: два концентратора, один был раньше Северным мостом, а теперь называется "Хаб контроля памяти" ("Memory Controller Hub"), другой был Южный мостом и называется "Хаб контроля ввода/вывода" (I/O Controller Hub). Функции концентраторов не поменялись, однако они стали более независимы, а интерфейс связи друг с другом представляет собой связь "один-к-одному" (point-to-point) через специальный концентратор (раньше мос-

ты соединялись через шину PCI, имеющую вдвое меньшую скорость, чем концентратор).

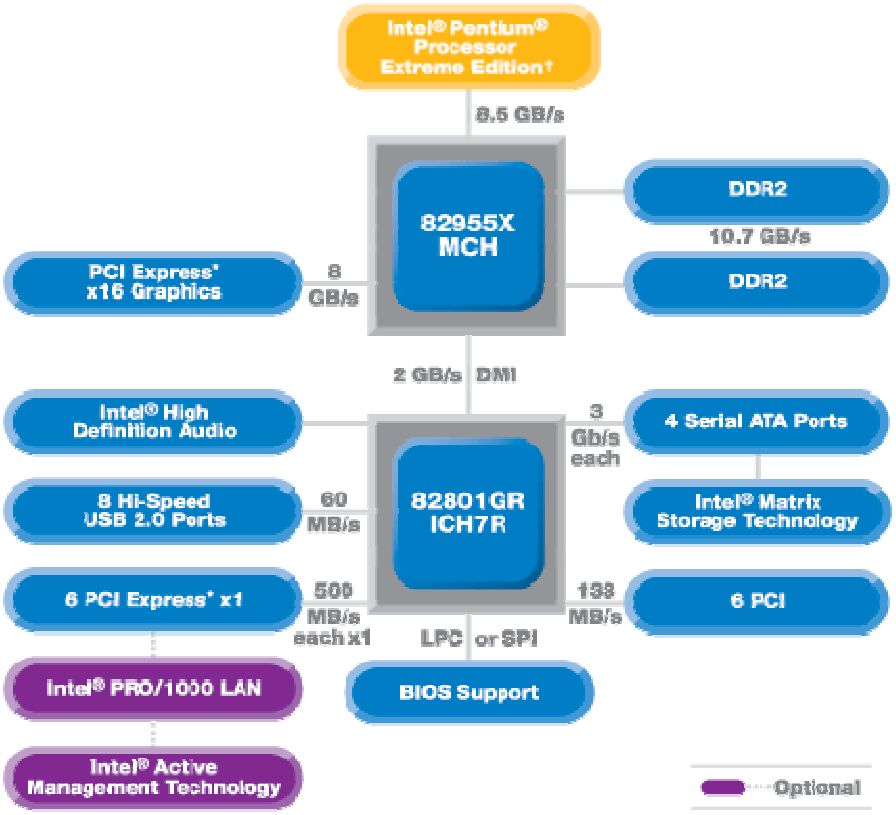


Рис. 12. Набор микросхем системной логики (Intel 945)

Все устройства ПК, включая мосты, взаимодействуют между собой по принципу "общей шины", т.е. через один общий канал – шину, представляющую собой совокупность наборов информационных, адресных сигналов и сигналов управления (шина адреса, шина данных и шина управления). При этом в один момент времени передавать информацию по шине может только одно устройство, а все остальные подключенные к ней устройства могут только принимать или игнорировать данные. Количество одновременно передаваемых сигналов называется разрядностью шины. Данные по шине передаются в виде цифр через равные промежутки времени. Для передачи единичного бита данных в определенный интервал времени посылается сигнал напряжения высокого уровня (обычно это +5В), а для передачи нулевого бита данных - сигнал напряжения низкого уровня (обычно это 0В). Чем больше линий в шине (т.е. чем больше её разрядность), тем больше битов можно передать за одно и тоже время. Таким образом, *скорость передачи данных по шине* (или её пропускная способность) определяется как произведение её тактовой частоты на разрядность.

Аналогом компьютерной шины можно называть обыкновенную электрическую сеть, в которой передающим устройством является электростанция, а принимающими устройствами являются все подключенные к сети устройства.

На самом деле в современных ПК не одна шина, а несколько:

- шина процессора;
- шина памяти;
- шина адреса;

- шины для подключения внешних устройств.

Когда говорят о шине, обычно имеют в виду последнюю (шину для подключения внешних устройств). Шина процессора соединяет его с северным мостом. Шина адреса фактически является частью шины процессора и используется для указания устройства, с которым происходит взаимодействие и адреса оперативной памяти. Шина памяти предназначена для передачи данных между оперативной памятью и устройствами ПК через северный мост.

Тактовая частота, с которой работает шина процессора, называется **системной** и соответствует внешней частоте, на которой работает процессор. Внутренние устройства процессора работают на внутренней частоте, которая получается путем умножения системной частоты и может превосходить её в несколько раз.

Название наборов микросхем системной логики обычно происходит от названия фирмы производителя и маркировки основной микросхемы (северного моста) - i810,1810E, i440BX,I820,VIA Apollo pro 133A, SiS630, UMC491,182C437VX и т.п. При этом используется только код микросхемы внутри серии: например, полное наименование SiS471 - SiS85C471. Последние разработки используют и собственные имена; в ряде случаев это - фирменное название (INTEL, VIA, Viper).

2.4. Шины и гнезда для подключения внешних устройств

Все периферийные устройства подключаются к материнской плате через специальные разъемы (см. рис. 13,14), которые условно можно разделить на внешние и внутренние. Условность такого разделения объясняется тем, что некоторые внутренние разъемы, используя специальные технические средства (кабели, планшеты и т.п.), могут стать внешними. Название разъема совпадает с названием интерфейса (шины), через которую будет передаваться информация между устройствами.

ISA (Industry Standard Architecture - архитектура промышленного стандарта) - основная шина на компьютерах типа PC AT. Другое название это шины - AT-Bus. Разрядность 8 или 16 бит. Частота передачи данных 8 Мб/с. Максимальная пропускная способность 16 Мбайт/с.

EISA (Enhanced ISA - расширенная шина ISA) - функциональное и конструктивное расширение ISA. Внешне разъемы имеют такой же вид, как и ISA, и в них могут вставляться платы ISA, но в глубине разъема находятся дополнительные ряды контактов EISA, а платы EISA имеют более высокую ножевую часть разъема с двумя рядами контактов, расположенных в шахматном порядке: одни чуть выше, другие чуть ниже. Разрядность 32 бита, работает также на частоте 8 МГц. Предельная пропускная способность - 32 Мбайт/с.

VLB (VESA Local Bus - локальная шина стандарта VESA) - 32-разрядное дополнение к шине ISA. Конструктивно представляет собой дополнительный разъем (116-контактный) при разъеме ISA. Разрядность 32 бита, тактовая частота в диапазоне от 25 до 50 МГц.

PCI (Peripheral Component Interconnect - соединение внешних компонент) - является дальнейшим шагом в развитии VLB. Разрядность - 32 (расширенный

вариант - 64) бита. Тактовая частота - до 33 МГц (PCI версии 2.1 - до 66 МГц). Пропускная способность шины до 528 Мбайт/с для 64-разрядной шины на 66 МГц.

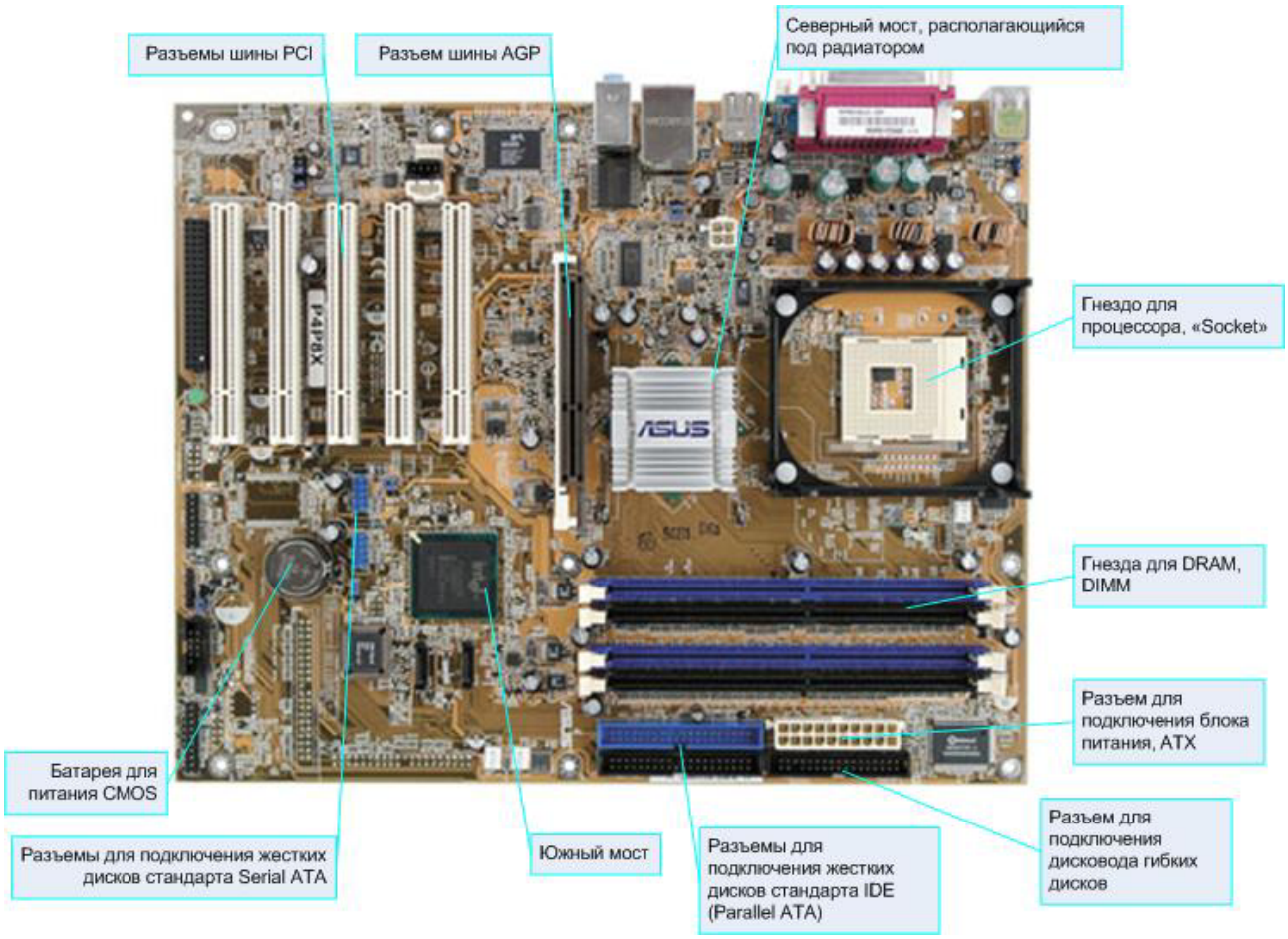


Рис. 13. Элементы материнской платы на примере ASUS P4P8X

Рис. 14. Внешние разъемы на примере материнской платы ASUS P4P8X

AGP (Accelerated Graphic Port - Ускоренный Графический Порт) - является дальнейшим развитием PCI нацеленным на ускоренный обмен графическими адаптерами с оперативной памятью. Пропускная способность увеличена за счет разрядности, тактовой частоты и способа передачи данных по шине.

Последовательный порт (COM). Термин «последовательный» означает, что передача данных осуществляется побитно, используя один проводник. Такой тип связи характерен для телефонной сети, в которой для передачи данных в одном направлении используется один проводник. Управление последовательным портом осуществляется контроллером UART (англ. Universal Asynchronous Receiver/Transmitter), который преобразует информацию из параллельного формата, используемого в ПК, в последовательный вид и наоборот. Термин «асинхронный» означает, что при передаче данных не используются никакие тактовые (синхронизирующие) сигналы и передача данных может происходить через произвольные интервалы. Для того чтобы определить границы передаваемого блока информации, используются стартовый и стоповый сигналы (т.е. определенная последовательность нулей и единиц). Для подключения устройств используются 9-ти или 25-ти штырьковые разъемы. Скорость обмена до 115 Кбит/с.

Параллельный порт (LPT). Информация через такой интерфейс передаётся побайтно в параллельном режиме, т.е. для передачи данных в одну сторону применяются восемь проводников. Первоначально LPT был разработан для подключения печатающих устройств – принтеров. Подключение осуществляется с использованием 25-ти штырькового разъема. Существуют одно- и двунаправленные параллельные интерфейсы.

PS/2 порты. Практически полный аналог COM порта. Служит для подключения клавиатуры или манипулятора мышь.

Универсальная последовательная шина (USB, Universal Serial Bus). Является развитием последовательного интерфейса. Разрабатывалась для того, чтобы стало возможным подключать несколько устройств к одному порту и делать это без отключения ПК.

FDD (Floppy Disk Drivers - накопитель на Гибких Магнитных Дисках) Конструктивно представляет собой 12x2 контактный игольчатый разъем с возможностью подключения двух дисководов через соединительный кабель – шлейф. В один момент времени информацию может передавать только один дисковод.

IDE (Integrated Drive Electronics) или ATA (AT Attachment). Используется для подключения устройств хранения информации (жесткого диска, CD-ROM и т.п.) Конструктивно представляет собой 20x2 контактный игольчатый разъем, к которому через шлейф можно подключить до 2-х дисковых устройств. Чаще всего на материнской плате устанавливают 2 IDE контролера: Primary и Secondary. Существуют также несколько протоколов обмена данными: UDMA/33 - ЗЗМБ/сек и UDMA/66 - 66МБ/сек. Протокол UDMA/66 обладает вдвое большей скоростью передачи данных за счет того, что данные передаются по обоим фронтам тактирующего сигнала в отличие от UDMA/33, вследствие чего необходим шлейф, в котором бы отсутствовали помехи от 2х параллельно идущих проводников. Для решения этой проблемы применяется 80-жильный шлейф, каждый второй проводник которого соединен с общим проводом для уменьшения помех.

2.5. Гнезда для подключения процессоров

Для установки процессоров на системную плату используются гнезда, которые в общем случае можно разделить на два типа: горизонтального исполнения, или сокет (англ. socket) и вертикального исполнения, или слот (англ. slot). В зависимости от того, какое гнездо установлено на материнскую плату, определяется перечень поддерживаемых процессоров.

Гнезда горизонтального типа представляют собой прямоугольный пластмассовый планшет с отверстиями, в которой располагаются металлические разъемы. Число отверстий зависит от типа сокета. Первые версии таких гнезд обозначались номерами от 1 до 8. Сейчас применяют нумерацию, соответствующую числу отверстий в гнезде – 370, 478 и т.д. Например, гнездо типа Socket-370 имеет 370 отверстий.

Для удобства установки и извлечения процессоров из гнезд типа “сокет” был разработан механизм, названный ZIF (англ. zero input force – установка без усилия). Суть этого механизма заключается в следующем. Гнездо состоит из двух расположенных друг над другом горизонтальных пластин. Нижняя пластина закрепляется на материнской плате. Верхняя пластина может перемещаться (скользить) по нижней пластине. При установке процессора пластины и, соответственно, разъёмы раздвигаются (используя рычаг, расположенный рядом с гнездом), процессор помещается в гнездо, затем пластины сдвигаются, зажимая в разъемах выводы процессора. При извлечении процессора процедура повторяется в обратном порядке. До появления ZIF-механизма процессор приходилось вставлять в гнездо, используя некоторое усилие, и извлечь его можно было только с использованием специальных приспособлений.

Гнездо типа «слот» конструктивно представляет пластиковый разъем с двумя рядами контактов. Процессоры в него устанавливаются вертикально. Для крепления процессоров чаще всего рядом с разъёмом устанавливают вертикальные стойки.

2.6. Запоминающие устройства (память)

В современных ПК для хранения информации используются следующие виды запоминающих устройств:

- динамическое запоминающее устройство с произвольным доступом (англ. dynamic random access memory – DRAM).
- статическое запоминающее устройство (англ. static random access memory – SRAM);
- постоянное запоминающее устройство (англ. read only memory - ROM);
- внешние устройства хранения данных (дисководы гибких дисков, жесткие диски, CD-ROM, стримеры и т.п.).

Первые два типа памяти называются энергозависимыми, так как при отключении ПК информация в них теряется. Для долговременного хранения информации используются носители третьего и четвертого типов.

Статическая и динамическая память используется для хранения оперативной информации.

Ячейками в динамической памяти являются крошечные конденсаторы. Наличие или отсутствие заряда определяет содержимое ячейки – 1 или 0. Конденсаторы не могут удерживать заряд бесконечно, поэтому в динамической памяти содержимое ячеек должно постоянно регенерироваться (т.е. перезаписываться). Регенерация происходит под управлением контроллера памяти, которые через равные промежутки времени (например, 15 мс) перечитывает содержимое ячеек памяти и заряжает конденсаторы заново. К сожалению, регенерация памяти требует некоторого времени, в течение которого доступ к памяти невозможен.

В статической памяти для организации ячеек используются специальные устройства – триггеры (построенные только на транзисторах), которые могут сохранять свое содержимое бесконечно долго, пока не будет выключено электропитание. За счет отсутствия циклов регенерации и высокой скорости доступа к ячейкам статическая память значительно быстрее, чем динамическая. Однако триггеры значительно дороже и больше по размерам, чем конденсаторы. Статическая память используется для организации буферов между быстродействующим процессором и медленной оперативной памятью.

Динамическая память оформляется в виде модулей, вставляемых в специальные разъемы на материнской плате. Статическая память чаще всего выполняется в виде одной или нескольких микросхем, располагаемых прямо на материнской плате. Кроме этого, современные процессоры содержат внутри статическую память малого размера.

На сегодняшний день существуют три типа разъемов для динамической памяти: SIMM (Single Inline Memory Module – одинарный модуль памяти), DIMM (Dual Inline Memory Module – двойной модуль памяти), RIMM (Rambus Interface Memory Module – модуль памяти с интерфейсом Rambus). Каждый из них, в свою очередь, имеет несколько разновидностей, определяющих способы функционирования памяти.

Модуль памяти типа SIMM имеет один ряд контактов, расположенных с двух сторон. В разъем модуль памяти вставляется под углом и поворачивается до вертикального положения, зажимаясь при этом держателями, расположенными по краям разъема. SIMM бывают двух видов, различающихся по числу выводов - 30 или 72. Чаще всего модули SIMM используются парами.

В модулях памяти типа DIMM контакты расположены также с 2-х сторон, но гальванически разделены между собой. В разъём модуль памяти вставляется вертикально с небольшим усилием, также зажимаясь расположенными по краям держателями.

Модули памяти типа RIMM визуально похожи на DIMM, хотя они обычно чуть шире. Микросхемы на модуле RIMM располагаются под углом. В отличие от модулей SIMM и DIMM, у которых объем памяти кратен степени числа 2, модули RIMM могут иметь любой размер. Для определения границы памяти используется специальная микросхема-заглушка. Она должна устанавливаться во все свободные слоты RIMM.

Для идентификации типа памяти, установленного в разъем, в состав каждого модуля входит специальная микросхема, хранящая всю необходимую техническую информацию.

2.7. Блок питания персонального компьютера

Назначение блока питания – преобразование электрической энергии, поступающей из сети переменного тока, в энергию, пригодную для питания узлов персонального компьютера. Входным может быть переменное напряжение с характеристиками 220В/50Гц или 120В/60Гц. Электронными схемами используется постоянное напряжение в +3.3В, ±5В и ±12В.

С внешней стороны (Рис. 15) блок питания имеет разъемы для кабеля, подключаемого к электрической сети и для кабеля, подключаемого к другим внешним (по отношению к системному блоку) устройствам, например монитор, звуковые динамики и т.п. Второй разъем может отсутствовать.

Для подключения к системной плате и периферийным устройствам блок питания с внутренней стороны оснащён соответствующими разъемами (см. рис. 16). Вид разъема определяет напряжение, которое подается от блока питания на его клеммы.

Рис. 15. Блок питания ПК

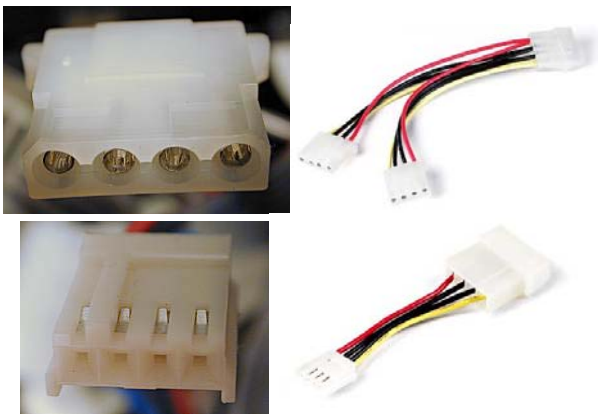


Рис. 16. Разъемы для подключения внутренних устройств

Для включения блока питания используется одно из двух:

- двухпозиционный выключатель, располагаемый на лицевой стороне корпуса (стандарт AT) и подключаемый к блоку питания напрямую;
- сигналы, поступающие от материнской платы по специальным каналам (стандарт ATX, NLX), которые генерируются либо устройствами ПК, либо при нажатии на выключатель, который подключается к специальному разъему на материнской плате и также располагается на лицевой стороне корпуса.

В первом случае для подключения к материнской плате используется двенадцати штырьковый разъем, во втором случае – двадцати штырьковый. В блоках питания с форм-фактором ATX дополнительно может быть установлен выключатель для того, чтобы исключить возможность включения блока питания по сигналу от материнской платы (см. рис. 9 справа внизу под разъёмом для кабеля питания).

ГЛАВА 3. ПРОЦЕДУРА ЗАГРУЗКИ ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА

Первым действием, которое выполняет компьютер при включении питания, является процедура **загрузки**, т.е. последовательность действий аппаратной части ПК по проверке состава (наличия или отсутствия тех или иных устройств) и запуска операционной системы.

Для современных персональных компьютеров с архитектурой IBM PC, построенных на основе семейства процессоров Intel (или совместимых с ним), процедура загрузки в общем случае выглядит следующим образом.

После нажатия кнопки «*Power»* источник питания выполняет самотестирование. Если все напряжения соответствуют номинальным величинам, то спустя некоторое время (примерно 0,1 - 0,5 с) он выдаёт на материнскую плату сигнал «*PowerGood*», после получения которого специальный триггер, вырабатывающий сигнал «RESET», снимает его с соответствующего входа микропроцессора. Далее сегментные регистры и указатель команд процессора устанавливаются в следующие состояния: CS = FFFFh; IP = 0; DS = SS = ES = 0. Все биты управляющих регистров и регистры арифметико-логического устройства устанавливаются в нулевое значение.

С момента снятия сигнала RESET микропроцессор начинает работу в реальном режиме, и, в течение примерно 7 циклов синхронизации, приступает к выполнению инструкции, считываемой из ROM BIOS, располагаемой по адресу FFFF:0000 (см. выше устанавливаемые значения регистров процессора). В этой области памяти содержится только команда перехода на реально исполняемый код BIOS. В этот момент процессор не может выполнять никакую другую последовательность команд, поскольку нигде в любой из областей памяти, кроме BIOS, её просто не существует.

Последовательно выполняя команды BIOS, процессор реализует функцию начального самотестирования POST (Power-On Self Test). На данном этапе тестируются процессор, память и системные средства ввода/вывода, а также производится конфигурирование программно-управляемых аппаратных средств материнской платы. Обнаружив ошибку, система определённым образом подаст звуковой сигнал.

В поисках встроенного драйвера видеоадаптера BIOS проверяет адреса памяти, начиная с 0000:0000 и заканчивая 0780:0000 (по умолчанию именно здесь должен располагаться такой драйвер). Если драйвер найден, проверяется контрольная сумма его кода, и, в случае совпадения с заданным значением, видеоадаптер инициализируется и выводится на экран курсор. В противном случае на экране появляется сообщение вида «C00 ROM Error». Если встроенный драйвер видеоадаптера не найден, то используется видеодрайвер, записанный в ПЗУ материнской платы, который пытается стандартным образом инициализировать видеоадаптер и вывести на экран курсор. Если и это не срабатывает, то видеоадаптер считается неисправным и подается соответствующий звуковой сигнал.

Далее сканируется память по адресам с C800:0000 по DF80:0000 с шагом 2 Кбайт в поисках встроенных драйверов любых других подключенных к материнской плате устройств (например, сетевых карт, модемов и т.п.). Обнаруженные драйверы выполняются так же, как и драйвер видеоадаптера. При несоответствии контрольных сумм выводится сообщение XXXX ROM Error, где XXXX - сегментный адрес некорректного драйвера.

После инициализации всех устройств BIOS проверяет значение слова по адресу 0000:0472, в котором содержится информация, корректирующая процесс дальнейшей проверки системы. Если здесь записано значение 1234h, то дальнейшая проверка устройств (включая оперативную память) не производится. Это возможно только в случае «горячей» перезагрузки ПК (например, при нажатии комбинации клавиш CTRL+ALT+DEL). В обычном режиме или при «холодной» перезагрузке (т.е. при нажатии клавиши RESET) здесь содержится значение 0000h.

В случае горячей загрузки, BIOS проверяет остальные подключенные устройства. Часть конфигурирования выполняется однозначно, а другая часть может определяться положением перемычек (переключателей) системной платы или содержимым энергонезависимой памяти CMOS. Для изменения CMOS используется утилита называемая «Setup», встроенная в BIOS.

Утилита Setup имеет интерфейс в виде меню или отдельных окон, иногда даже с поддержкой графики и мыши. Для запуска Setup во время выполнения POST появляется предложение нажать определённую комбинацию клавиш, например DEL.

После завершения POST BIOS определяет порядок поиска (boot sequence) внешних устройств, чтобы загрузить операционную систему (специальную программу, управляющую работой ПК). Этот порядок определяется одним из параметров, содержащихся в CMOS. Например, если последовательность определена как A, C, D, то сначала будет проверен дисковод и, если в нём находится дискета, BIOS попытается использовать её для загрузки ОС. Если дискета не обнаружена, то будет проверен диск C, затем D.

После того, как определено устройство, с которого будет происходить загрузка операционной системы, BIOS считывает с него информацию, располагаемую в самом начале, в оперативную память по адресу 0000:7C00. После чего проверяется, является ли это информация программой дальнейшей загрузки операционной системы. Если значения первых байтов считанного блока данных некорректные, на экране отображается сообщение об ошибке загрузочной записи и производится проверка следующего диска в списке. Если ни на одном из указанных носителей нет загрузочной программы, то на экран выводится сообщение об ошибке и загрузка системы останавливается.

С этого момента начинается загрузка операционной системы, процедура которой зависит от её типа.

ГЛАВА 4. ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В ЭВМ

Любая информация в ЭВМ, включая текст, аудио и видео, представляется в виде последовательности нулей и единиц. Используя определённые правила, двоичные разряды формируются в числа, которые, в свою очередь, преобразуются в нужную для человека форму (текст, звук, изображение и т.п.). Важным этапом в процессе изучения принципов работы ЭВМ является понимание того, каким же образом внутри них хранится информация.

4.1. Краткая история возникновения чисел, алгебры и систем счисления

Потребность человечества в числах определялась необходимостью счёта и измерения, возникавшей в непосредственной практической деятельности. Затем число становится основным понятием математики, и дальнейшее развитие этого понятия определяется потребностями этой науки.

Первоначально числа обозначались чёрточками на материале, служащем для записи (папирус, глиняные таблички и т.д.). Числа соотносились с конкретными предметами, например три камня, четыре палочки и т.п. Арифметические операции являлись действиями по объединению нескольких совокупностей предметов в одну или разделения одной совокупности на части. Лишь в многовековом опыте сложилось представление об отвлечённом характере этих действий, о независимости количественного результата действия от природы предметов, составляющих совокупности, о том, что, например, два предмета и три предмета составят пять предметов независимо от природы этих предметов. Тогда стали разрабатывать правила действий, изучать их свойства, создавать методы для решения задач, т. е. начинается развитие науки о числах — арифметики.

Со временем потребность человека в арифметических операциях возрастала. Для их выполнения были разработаны различные приспособления, одними из которых являются электронно-вычислительные машины.

4.1.1. Системы счисления

Для записи чисел человечеством придуманы различные правила, называемые *системами счисления*. По этим правилам любое число представляется в виде набора специальных символов – *цифр* (от араб. сифр — нуль, буквально — пустой; арабы этим словом называли знак отсутствия разряда в числе). Получение количественного эквивалента числа осуществляется по *алгоритму замещения*, согласно которому сначала цифры заменяются их количественными эквивалентами, а затем эквивалент числа получается путем арифметических операций над эквивалентами цифр.

В зависимости от того, меняет ли свое количественное значение цифра при разном положении в числе, системы счисления можно классифицировать на *непозиционные* и *позиционные* системы.

В *системах счисления первого типа (непозиционных)* число образуется из цифр, значение которых не изменяется при различном положении цифр в числе. Примером таких систем служит *римская* система счисления. В ней в качестве

цифр для составления чисел используются буквы латинского алфавита, I – означает единицу, V – пять, X – десять, L – пятьдесят, C – сто, D – пятьсот, M – тысяча. Для получения числа требуется просто просуммировать количественные эквиваленты входящих в него цифр, с учетом того, что если младшая цифра идет перед старшей цифрой, то она входит в сумму с отрицательным знаком. Например, DLXXVII = пятьсот + пятьдесят + десять + десять + пять + один + один = пятьсот семьдесят семь. Или CDXXIX = минус сто + пятьсот + десять + десять + минус один + десять = четыреста двадцать девять.

В *позиционных системах счисления* количественное значение цифры определяется её позицией в числе. Номер позиции называется *разрядом*. Число цифр, используемых для представления чисел, называется *основанием*. Количественное значение числа в позиционной системе счисления, состоящего из n цифр $\{a_i\}$, $i \in \overline{0,n-1}$ (т.е. числа, имеющего вид $a_{n-1}a_{n-2}\ldots a_1a_0$), может быть получено следующим образом:

$$A_{(p)} = a_{n-1}p^{n-1} + a_{n-2}p^{n-2} + \ldots + a_1p^1 + a_0p^0, \qquad (1)$$

Число n называется разрядностью, и определяет максимальный эквивалент, который можно получить для такого числа:

$$A_{(p)}^{\max} = p^n.$$

В силу того, что ЭВМ строится на базе логических схем, которые могут иметь только два состояния – включено и выключено, то все числа в них представлены в *двоичной системе счисления*, которая по своей сути является позиционной. Набор цифр в этой системе состоит из 0 и 1 (основание равно 2). Например, число в двоичной системе может иметь вид $10100111_{(2)}$. Количественный эквивалент такого числа равен ста шестидесяти семи ($167_{(10)}$).

Очевидно, что для человека выполнение арифметических операций с двоичными числами затруднительно и не естественно. Поэтому для ввода и вывода чисел используются другие системы счисления. Наибольшее распространение получили восьмеричная, десятичная и шестнадцатеричная системы счисления и представление целых чисел в двоично-десятичной форме (BCD, Binary coded decimal).

В *десятичной системе счисления* набор цифр включает 0, 1, 2, 3, 4, 5, 6, 7, 8 и 9. Например, число $10_{(10)}$ имеет количественный эквивалент десять ($1\cdot10^1 + 0\cdot10^0$). Эта система используется нами в повседневной жизни. Такое распространение она получила из-за того, что первоначально счет предметов производился с использованием пальцев на человеческих руках. Как известно, у обычного человека на руках ровно десять пальцев.

В *восьмеричной системе счисления* набор цифр включает 0, 1, 2, 3, 4, 5, 6 и 7. Например, число $77_{(8)}$ имеет количественный эквивалент шестьдесят три ($7\cdot8^1 + 7\cdot8^0$), а $10_{(8)}$ равно восьми ($1\cdot8^1 + 0\cdot8^0$).

В *шестнадцатеричной системе счисления* набор цифр включает римские цифры от 0 до 9 и 6 букв латинского алфавита – A (десять), B (одиннадцать), C (двенадцать), D (тринадцать), E (четырнадцать), F (пятнадцать). Например, чис-

ло $FF_{(16)}$ имеет количественный эквивалент двести пятьдесят пять $(15 \cdot 16^1 + 15 \cdot 16^0)$, а $100_{(16)}$ равно двести пятьдесят шесть $(1 \cdot 16^2 + 0 \cdot 16^1 + 0 \cdot 16^0)$.

Двоично-десятичные числа – это специальный вид представления числовой информации, в основу которого положен принцип кодирования каждой десятичной цифры группой из четырёх бит. При этом каждый байт содержит одну или две цифры. Первый способ называется *неупакованное число*, второй - *упакованное*. Например, число 3456 может быть записано как неупакованное в виде: $00000011\ 00000100\ 00000101\ 00000110_{(2)}$, или в упакованное $00110100\ 01010110_{(2)}$.

4.1.2. Перевод чисел из одной системы счисления в другую

Для выполнения арифметических операций над числами они должны быть представлены в одной системе счисления.

Перевод чисел из *любой системы счисления в десятичную систему* осуществляется простым получением их количественных эквивалентов и записи в виде десятичного числа.

Перевод чисел из *десятичной системы в любую другую* осуществляется путем деления исходного числа на основание требуемой системы и записи остатков от деления в обратном порядке. Например, смотри рис. 17.

Перевод числа из шестнадцатеричной системы в двоичную систему может быть осуществлен путем представления каждой его цифры в виде двоичной тетрады, и последовательной записи этих тетрад. Например, число $FF_{(16)}$ представляется как $1111\ 1111_{(2)}$. А число $3A_{(16)}$, как $0011\ 1010_{(2)}$. Соответственно, перевод числа из двоичной системы в шестнадцатеричную систему осуществляется путем деления его на тетрады, и представления каждой тетрады в виде шестнадцатеричной цифры. Например, $10111001_{(2)}$, представляется как $1011\ 1001_{(2)}$, и в шестнадцатеричной системе имеет вид $B9_{(16)}$.

Перевод числа из восьмеричной системы в двоичную систему осуществляется аналогично переводу числа из шестнадцатеричной системы, только цифры заменяются не тетрадами, а двоичными триадами. Например, $77_{(8)}$ будет представлено как $111\ 111_{(2)}$, а число $10_{(8)}$, как $001\ 000_{(2)}$. Двоичное число при переводе в восьмеричную систему счисления сначала делится на триады, а затем каждая триада представляется в виде восьмеричной цифры. Например, $11100111_{(2)}$, делится на $011\ 100\ 111_{(2)}$, и получается $347_{(8)}$.

Перевод чисел из шестнадцатеричной системы счисления в восьмеричную систему и наоборот осуществляется в два этапа. Сначала исходное число переводится в двоичную или десятичную систему, а затем из двоичной или десятичной системы число переводится в требуемую систему счисления. Например, число $FF_{(16)}$ в двоичной системе имеет вид $1111\ 1111_{(2)}$, и в восьмеричной системе оно примет вид $377_{(8)}$.

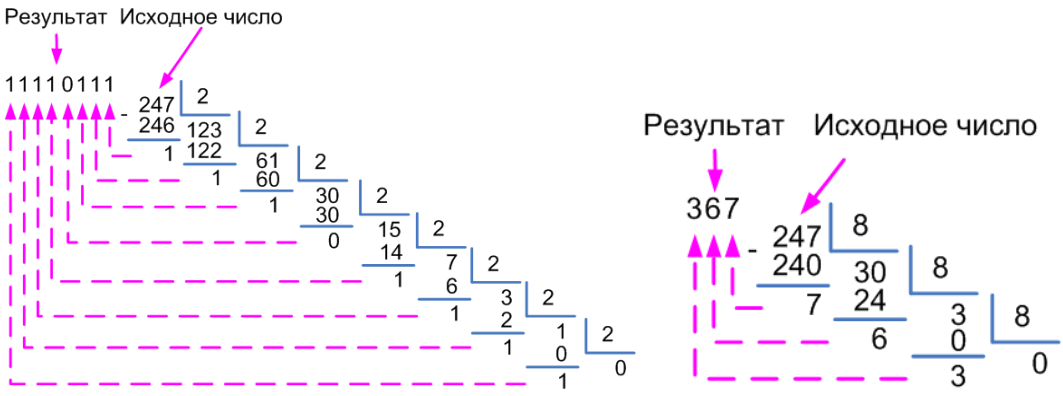


Рис. 17. Перевод числа 247 из десятичной системы счисления в двоичную(слева) и восьмеричную (справа) системы

4.1.3. Представление отрицательных целых и вещественных чисел в ЭВМ

Отрицательные целые числа в ЭВМ представляются в специальном виде, называемом *дополнительным кодом*, который позволяет при выполнении операций исключить отличия отрицательных от положительных чисел.

Дополнительный код отрицательного числа представляет собой результат инвертирования (замены в числе нулей на единицы и наоборот) каждого бита двоичного числа (модуля отрицательного числа) и прибавления к нему единицы.

Обратное преобразование числа из дополнительного кода в обычный вид осуществляется аналогичным образом (сначала инвертируется, затем прибавляется 1).

Например, рассмотрим число $-185_{(10)}$. Его модуль в двоичном виде имеет вид - $10111001_{(2)}$. Чтобы его перевести в дополнительный код, надо произвести его инвертирование. Причем инвертируется вся ячейка памяти, отведённая под число (будем считать, что мы работаем с 16-ти разрядными ячейками). В результате получится число $1111111101000110_{(2)}$. Добавляя к нему единицу, получаем число в дополнительном коде $1111111101000111_{(2)}$.

Если требуется определить, является ли число отрицательным, то необходимо проанализировать его первый разряд. Если он равен 1, то число отрицательное, если 0 – то положительное. Если считать, что результат предыдущего примера только положительный, то в десятичном виде он равен 65351. Именно поэтому изменяется диапазон отрицательных чисел, которые можно записать в n-разрядную ячейку. Например, в 8-разрядную ячейку можно записать целые положительные числа из диапазона от $0_{(10)}$ до $255_{(10)}$, а целые отрицательные из диапазона $-128_{(10)}$ до $127_{(10)}$.

Вещественные числа в ЭВМ могут быть представлены в форме с фиксированной или плавающей запятой.

Представление числа *в форме с фиксированной запятой*, которую иногда называют также *естественной формой*, включает в себя знак числа и его мо-

дуль в q-ичном коде. Здесь q - это *основание системы* или *база*. В ЭВМ чаще всего используется двоичная система, однако существуют ещё 8- и 16-ичные формы. Числам с фиксированной запятой соответствует запись вида $a_{n-1}a_{n-2}\ldots a_1a_0a_{-1}a_{-2}\ldots a_{-m+1}a_{-m}$. При этом положение запятой никак не фиксируется, а лишь подразумевается в процессе выполнения арифметических операций. Точность числа в формате с фиксированной запятой определяется числом разрядов, отводимых под дробную часть.

Получить количественный эквивалент числа с фиксированной запятой можно по формуле:

$$A_{(p)} = a_{n-1}p^{n-1} + a_{n-2}p^{n-2} + \ldots + a_1p^1 + a_0p^0 + a_{-1}p^{-1} + a_{-2}p^{-2} + \ldots + a_{-m}p^{-m}.$$

Перевод из двоичной системы счисления в шестнадцатеричную осуществляется аналогично целым числам – деление на тетрады и представление каждой тетрады в виде шестнадцатеричной цифры и наоборот. Деление на тетрады осуществляется от запятой (влево для целой части и вправо для дробной).

Перевод числа с фиксированной запятой из десятичной системы в двоичную осуществляется в два этапа. На первом этапе переводится целая часть обычным образом. На втором этапе переводится дробная часть умножением её на 2 и выделением целой части на каждом шаге (см. рис. 18). Умножение производят до тех пор, пока не будет достигнута требуемая точность (будет получено требуемое количество разрядов после запятой) или при очередном умножении получается дробная часть равная нулю.

Вещественные числа в формате с плавающей запятой представляются в виде двух групп цифр – мантиссы и порядка. Число представляется в виде произведения $X = \pm mq^{\pm p}$, где m мантисса числа X, q - основание системы счисления, p - порядок числа. Форму записи чисел с плавающей запятой также называют *нормальной*. Точность числа в формате с плавающей запятой зависит от числа разрядов, отводимых под мантиссу и порядок.

Для представления чисел с плавающей запятой в ЭВМ был разработан стандарт IEEE 754, согласно которому выделяют два типа чисел: 32-битовое с 8-ю разрядами под порядок и 64-битовое с 11 разрядами под порядок. Первый тип называется одинарным или простым, второй – двойным или с двойной точностью.

4.1.4. Представление символьной информации в ЭВМ

Каждому символу однозначно сопоставляется некоторая двоичная последовательность, называемая его **кодом**. Совокупность возможных символов и их кодов образуют **таблицу кодировки**.

В настоящее время применяется огромное количество различных кодировок символов. Общим (хотя и не обязательным) для всех систем кодирования является **весовой принцип,** согласно которому коды цифр (т.е. двоичные числа) увеличиваются по мере увеличения цифры, а коды латинских букв увеличиваются в алфавитном порядке. Например, код символа ‘1’, на единицу меньше кода символа ‘2’, а код символа ‘b’ на единицу меньше кода символа ‘c’. Требований на порядок расположения специальных символов (символов нацио-

нальных языков, знаков препинания, математических символов и т.п.) обычно не предъявляется.

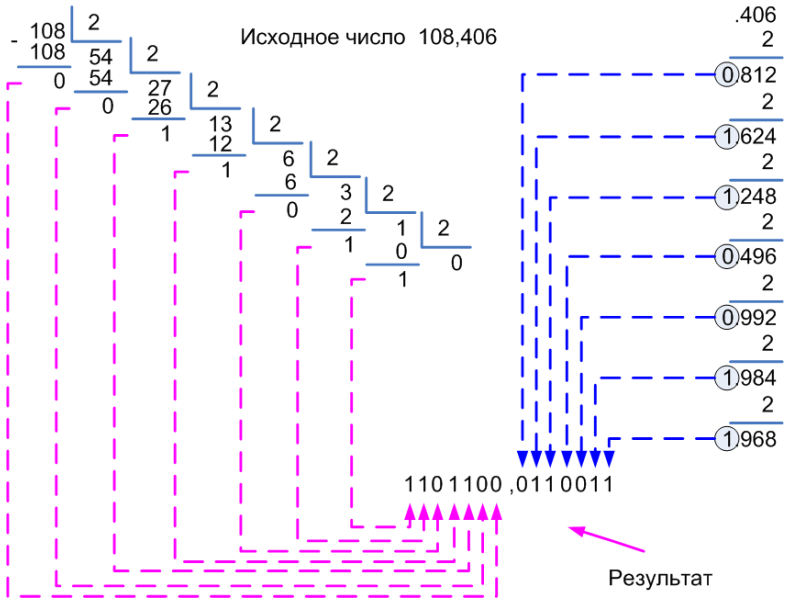


Рис. 18. Перевод числа 108,406 из десятичной системы в двоичную систему счисления

Самыми первыми и наиболее распространёнными являются кодировки, представляющие символы восьмиразрядными двоичными числами. При этом в таблице может содержаться не более 256 кодов различных символов. Примерами таких таблиц являются:

- расширенный двоично-кодированный код EBCDIC (Extended Binary Coded Decimal Interchange Code). Так же он известен под названием ДКОИ. Кодировка EBCDIC используется в ЭВМ, производимых фирмой IBM;
- американский стандарт кода для представления информации ASCII (American Standard Code for Information Interchange), разработанный Институтом стандартизации США (ANSI).

Таблица ASCII делится на две части – базовая и расширенная. Базовая таблица закрепляет значение кодов от 0 до 127 (т.е. использует только 7 бит), а расширенная относится к символам с номерами от 128 до 255. Изначально существовала только базовая часть таблицы ASCII, а старший (восьмой) бит использовался для контроля четности (т.е. принимал единичное значение, если в 7-и битной части четное число единиц). Основная таблица описывает 128 символов, из которых (см. рис. 19):

- первые 32 кода отданы производителям аппаратных средств (в первую очередь производителям печатающих устройств). В этой области размещаются так называемые управляющие коды, которым не соответствуют никакие символы языков, и эти коды не выводятся ни на экран, ни на устройства печати, но ими можно управлять тем, как производится вывод прочих данных;
- коды с 32 по 127 описывают символы английского алфавита, знаков препинания, цифр, арифметических действий и некоторых вспомогательных символов.

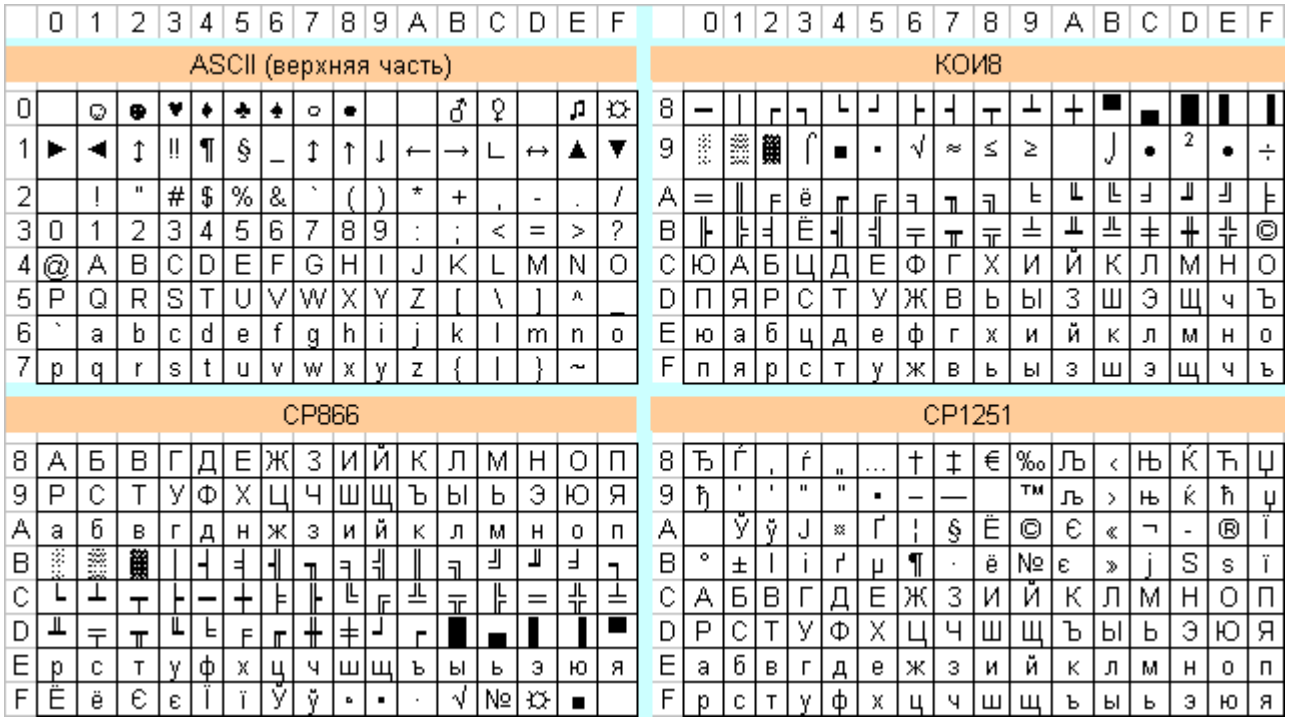


Рис. 19. Таблицы символов ASCII

Стандарт ASCII с 8 битами не определяет содержание верхней половины таблицы кодировки, которая используется разработчиками разных стран для представления символов соответствующих языков. Однако, для того, чтобы обеспечить единообразие правил представления символов этой части кодовой таблицы, Международная организация по стандартизации (ISO) взяла ответственность по определению семейства стандартов, известных как семейство ISO 8859-X. Это семейство представляет собой совокупность 8-ми битных кодировок, где младшая половина каждой кодировки (символы с кодами 0-127) соответствует ASCII, а старшая половина определяет символы для различных языков. В зависимости от использования кодов 128-255 различают следующие вариации стандарта ISO 8859 (см. табл. 1).

Таблица 1. Варианты стандарта ISO 8859-X

Стандарт	Характеристика
ISO 8859-0	Новый европейский стандарт (так называемый Latin-0)
ISO 8859-1	Языки западной Европы и Латинской Америки (Latin-1)
ISO 8859-2	Языки стран центральной и восточной Европы
ISO 8859-3	Языки стран южной Европы, мальтийский и эсперанто
ISO 8859-4	Языки стран северной Европы
ISO 8859-5	Языки славянских стран с символами кириллицы
ISO 8859-6	Арабский язык
ISO 8859-7	Современный греческий язык
ISO 8859-8	Языки иврит и индиш
ISO 8859-9	Турецкий язык

Продолжение таблицы 1

ISO 8859-10	Языки стран северной Европы (лапландский, исландский)
ISO 8859-11	Тайский язык
ISO 8859-13	Языки балтийских стран
ISO 8859-14	Кельтский язык
ISO 8859-15	Комбинированная таблица для европейских языков
ISO 8859-16	Специфические символы для языков: албанского, хорватского, английского, финского. Французского, немецкого, венгерского, ирландского, итальянского, польского, румынского и словенского

Несмотря на то, что в ISO разработали стандарты для представления языков многих стран, разработчики программного обеспечения предпочитают пользоваться другими (собственными) кодировочными таблицами.

Одной из альтернатив стандарту ISO 8859-5 стала кодовая таблица КОИ8, разработчики, которой поместили символы русской кириллицы в верхней части расширенной ASCII таблицы таким образом, что позиции кириллических символов соответствуют их фонетическим аналогам в английском алфавите в нижней части таблицы. Это означает, что если в тексте, написанном в КОИ8, убирать восьмой бит каждого символа, то текст останется читаемым, хотя и выглядеть будет забавно. Например, слова «привет», будет выглядеть как «privet». Такая кодировка широко используется (например, при передаче SMS-сообщений). Следует отметить, что KOI8-R подходит только для русских текстов, и как следствие был создан украинский вариант KOI8-U.

Компания Microsoft в своих операционных системах MS-DOS и MS-WINDOWS использует свои кодовые страницы, называемые OEM (Original Equipment Manufacturer) (см. табл. 2):

Таблица 2. Наиболее распространенные страницы OEM

CP437	США, страны западной Европы и Латинской Америки
CP708	Арабские страны
CP737	Греция
CP866	Российская кодировка для MS-DOS
CP932	Япония
CP936	Китай
CP1251	Российская кодировка для MS-Windows

К сожалению, стандарты ISO, КОИ8 и OEM хотя и предназначены для одного и того же (кодированию символов национальных языков), но не согласованы между собой. Поэтому, при передаче информации, необходимо четко оговаривать, с использованием какой таблицы она закодирована.

Стандарт ASCII (и все аналогичные ему стандарты) в силу 8-битной кодировки имеет существенные ограничения по числу символов, которые могут быть закодированы. По этой причине в 1993 году компаниями Apple Computer,

Microsoft, Hewlett-Packard, DEC и IBM был разработан стандарт ISO 10646, в котором символы кодируются 16 битами. Этот стандарт получил название **Unicode**.

Такой подход позволил создать кодировку, способную описать 65536 символов, то есть даёт возможность одновременно представлять символы всех «живых» и «мертвых» языков. Для букв русского алфавита выделены коды из диапазона 1040-1093.

4.1.5. Вывод символьной информации. Шрифты

При кодировании символьной информации в ЭВМ никак не описывается, как будет выглядеть каждый символ на экране или на бумаге. А только лишь определяются соглашения вида «если это число X, то будем понимать его как символ «буква а», а если это число Z, то будем понимать его как символ «запятая» и т.д. Для того, чтобы описать, как должны выглядеть символы на экране или на бумаге, используются дополнительные правила – **шрифты**, в которых для каждого числа однозначно определятся вид соответствующего символа.

Шрифты бывают *растровые* или *векторные*. В первом случае в памяти ЭВМ хранится образ символов (растр), который при необходимости выбирается из неё и выводится пользователю. Растр (см. рис. 20) – это матрица определённого размера, в которой в тех ячейках, которые должны быть закрашены, помещается 1, в остальных 0. Во втором случае в памяти ЭВМ хранятся команды, которые надо выполнить устройству отображения, чтобы вывести требуемый символ.

В текстовом режиме вся область для вывода информации поделена на ячейки, называемые **знакоместом**. В каждую ячейку может быть выведен только один символ.

4.2. Типы данных, используемые для хранения переменных в программах, написанных на языке Си

В стандарте языка Си определены три базовых типа переменных:

Тип	Описание
int	Знаковая целая переменная
char	Символьная переменная
float	Переменная, хранящая дробное число в формате с плавающей запятой

Каждый из типов определяет формат хранения данных в переменных и, соответственно, диапазон допустимых значений, которые эти переменные могут принимать. Для изменения формата хранения данных используются модификаторы типа, определяющие наличие или отсутствие в переменной знака (unsigned или signed), двойную точность при представлении чисел с плавающей запятой (double), увеличенный или сокращенный диапазон значений (long, long long или short).

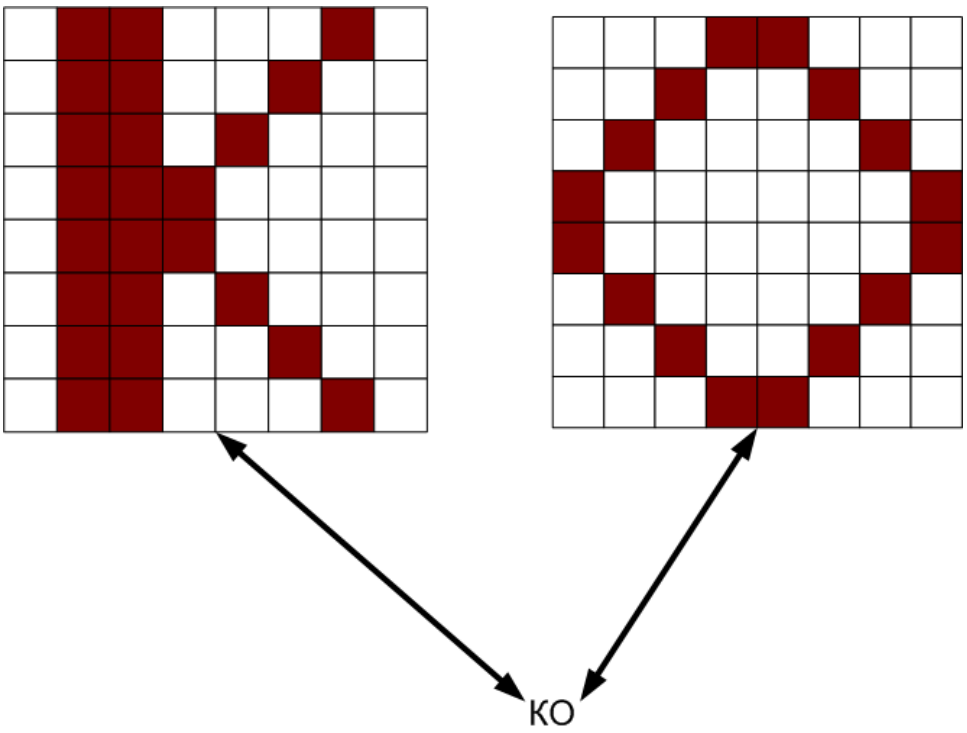


Рис. 20. Пример растрового шрифта

Для ввода и вывода значения переменных используются функции *scanf* и *printf* из стандартной библиотеки ввода/вывода. Чтобы определить, в каком виде выводить значения переменных, в функцию передаётся требуемый формат в виде последовательности символов % (процент) и буквы, определяющей необходимый формат:

%d – означает вывести (или ввести) целую знаковую переменную в десятичной системе счисления;

%o – целую переменную в восьмеричной системе счисления;

%x – целую переменную в шестнадцатеричной системе счисления;

%f – число с плавающей запятой и т.д.

Например, если функция *printf* вызвана следующим образом: *printf (“Значение x = %x\n”, x);*, то в стандартный поток вывода будет помещена фраза «Значение переменной x = », после чего туда же будет выведено значение переменной x в шестнадцатеричной системе счисления.

4.3. Разрядные и логические операции в языке Си. Маскирование

На практике часто приходится определять или задавать состояние устройств или управлять выполнением программы с использованием **двоичных флагов** – переменных, в которых может храниться только 0 или 1. Причем флаг считается установленным, если он содержит 1 и не установленным в противном случае. Примером использования флагов можно назвать программное управление выключателями: если необходимо выключатель перевести в состояние «включено», то устанавливаем флаг в 1, в противном случае в 0.

Конечно же, для представления флага можно использовать одну целую переменную. Однако, чаще всего приходится иметь дело одновременно с большим количеством флагов (управлять большим количеством выключателей). В

этом случае целесообразно в качестве флага использовать один разряд целой переменной. Таким образом, при использовании 32-х разрядных переменных, одной переменной может быть описано сразу 32 флага. Именно такой способ применяется в современных процессорах для описания их состояния (регистр флагов – ячейка памяти, содержащая несколько двоичных разрядов-флагов).

Для манипуляции отдельными битами целых переменных в языке Си используются **разрядные операции:** И, ИЛИ, НЕ, ИСКЛЮЧАЮЩЕЕ-ИЛИ, СДВИГ ВЛЕВО, СДВИГ ВПРАВО. Операция И обозначается символом &, операция ИЛИ - |, операция НЕ - ~, операция ИСКЛЮЧАЮЩЕЕ ИЛИ - ^, СДВИГ ВЛЕВО - <<, СДВИГ ВПРАВО - >>. Выполнение этих операций происходит в двоичной системе счисления, несмотря на то, в каком виде представлены её операнды. Например, операция 2 & 5 будет равна 0 (т.е. 010 & 101 = 0).

Чтобы получить значение определённого флага (т.е. располагающегося в определённом разряде числа) необходимо выполнить следующую последовательность действий:

flag = (registr >> (k – 1)) & 0x1,

где registr – это переменная, хранящая флаги, k – номер разряда (по порядку) в котором находится требуемый флаг. В результате этих действий переменная registr будет сдвинута вправо таким образом, что требуемый флаг окажется в первом разряде, после чего будет произведена операция логического умножения на единицу (т.е. все разряды числа, кроме первого, будут умножены на 0). В переменную flag будет помещено либо 1, либо 0, в зависимости от того, в каком состоянии был флаг.

Если требуется установить значение флага в единицу, то необходимо выполнить следующие действия:

registr = registr | (1 << (k – 1)).

Другими словами, необходимо выполнить операцию поразрядного ИЛИ между регистром флагов и числом, в котором в нужном разряде установлена 1, а в остальных разрядах числа содержатся нули.

Если требуется установить значение флага в ноль, то необходимо выполнить следующие действия:

registr = registr & (~(1 << (k – 1))).

Другими словами, необходимо выполнить операцию поразрядного И между регистром флагов и числом, в котором в нужном разряде установлен 0, а в остальных содержатся единицы. Такое число получается путем инвертирования числа, содержащего единицу в том разряде, в котором нам нужен 0. Такая операция называется **маскированием**, а число, определяющее разряд (второй операнд) – маской. Ясно, что выделение разряда и установка его в ноль выполняется одинаковым образом, с тем лишь отличием, что в первом случае используется маска с единицей в требуемом разряде и остальными разрядами равными нулю, а во втором случае – инверсная ей маска.

Очевидно, что одновременно можно работать с несколькими разрядами. Необходимо только подобрать соответствующим образом второй операнд (маску).

ГЛАВА 5. УСТРОЙСТВА ВВОДА-ВЫВОДА ИНФОРМАЦИИ. ТЕРМИНАЛЫ

5.1. Клавиатура

Для ввода информации в ЭВМ обычно используется устройство, называемое клавиатурой. В общем случае оно представляет собой набор переключателей – клавиш, расположенных в виде прямоугольной матрицы присоединенной к специальному процессору (см. рис. 21).

5.1.1. Общее устройство клавиатуры

Нажимая на клавишу, пользователь замыкает соответствующий переключатель, и, тем самым, на определённые входы процессора подаёт положительные сигналы (логические единицы). Процессор шифрует значения этих входов, получая цифровой номер нажатой клавиши, и передаёт это значение в ЭВМ. Цифровой номер клавиши называется её **скан-кодом**. Такое название принято вследствие того, что для определения нажатой клавиши процессор как будто сканирует значения своих входов, а лишь затем выполняет операцию шифрации.

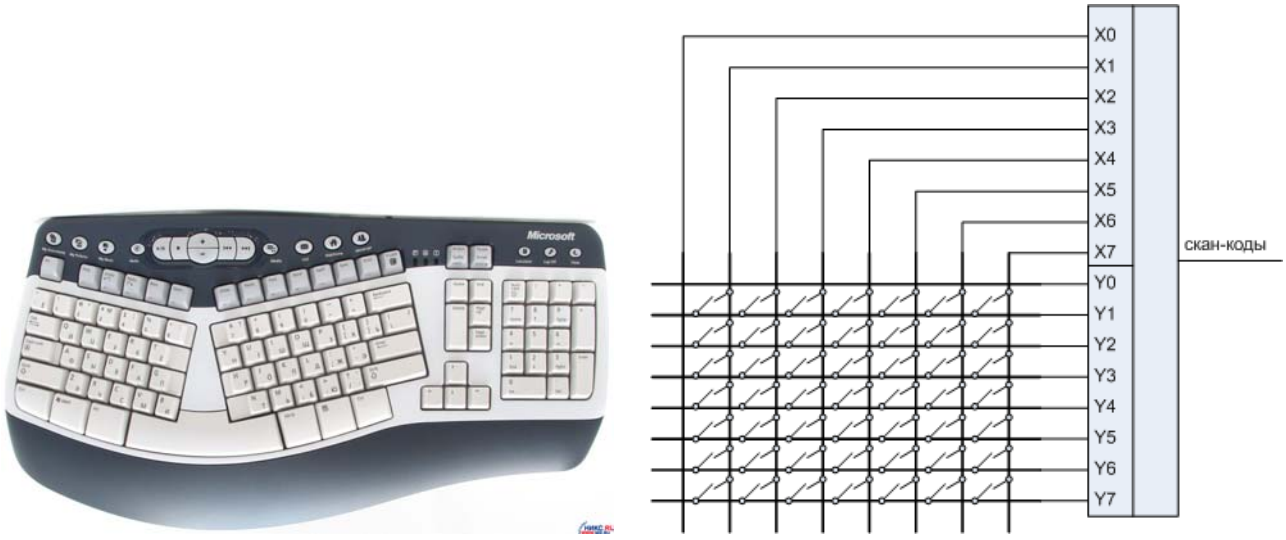


Рис. 21. Пример клавиатуры (слева) и её простейшая схема (справа)

Скан-код клавиши может состоять из одного или нескольких байтов (см. приложение). Обычно клавиши, на которых нанесены символьные обозначения, имеют однобайтовые коды, а коды управляющих клавиш – несколько байт. Таблица сопоставления кодов клавишей зависит от типа клавиатуры. В персональных компьютерах принято кодировать клавиши способом, приведённым в приложении.

Номер клавиши однозначно связан только с её местоположением в матрице (клавиатуре), и никак не зависит от нанесённых на неё обозначений. Например, скан-код 0x2B соответствует клавише с ASCII обозначениями “f”, “F”, “a”, “A”.

Обратите внимание (!!!), что скан-код клавиши, и код символа – это разные вещи. Одному скан-коду в зависимости от режима работы клавиатуры может соответствовать несколько символов или вообще может не соответствовать никакого символа, как, например, у управляющих клавиш.

Если пользователь продолжает держать клавишу нажатой и, соответственно, переключатель держать в замкнутом состоянии, то процессор после некоторого ожидания повторяет процедуру шифрации нажатой клавиши и передачи её кода в ЭВМ. Это сделано для того, чтобы при необходимости многократного нажатия на клавишу (например, для перемещения курсора на несколько строк или столбцов) пользователю не приходилось «долбить» по клавиатуре.

Когда пользователь отпускает клавишу, или размыкает переключатель, процессор так же сообщает об этом ЭВМ, используя определённое числовое значение. Обычно это значение равно значению, получаемому при нажатии клавиши, только с единицей в старшем бите.

После получения скан-кода клавиши ЭВМ (а точнее соответствующее программное обеспечение – драйвер клавиатуры или пользовательская программа) сопоставляет ей определённую символьную последовательность, называемую **кодом клавиши.** Очевидно, что правила сопоставления, зависят не только от того, какая клавиша нажата, но и от того, в каком режиме работает клавиатура. Например, вводится ли русский или английский текст, была ли нажата одна клавиша или их было нажато несколько, является ли нажатая клавиша символьной (т.е. на неё нанесен символ) или управляющей (например, клавиша перемещения курсора) и т.п.

По сути, эта последовательность является числовыми кодами символов, изображенных на соответствующей клавише. Для управляющих клавиш, у которых нет символьного изображения (например, клавиши управления курсором или функциональные клавиши) формируется последовательность из нескольких символов (байт), первым из которых идет специальный символ, сигнализирующий о том, что нажата специальная клавиша.

Для некоторых клавиш вообще не формируется символьные последовательности, а их нажатие приводит к тому, что ЭВМ переводит клавиатуру в новый режим. Например, клавиша Shift приводит к тому, что при её удержании символьные клавиши в дальнейшем будут сопоставляться заглавным буквам.

Часто кроме клавиш клавиатура оснащена дополнительными индикаторами, предназначенными для отображения её состояния или текущего режима работы. Например, нажатие клавиши NumLock вызывает изменение функциональных назначений цифровой части клавиатуры, о чем сигнализирует соответствующий светодиодный индикатор. Для изменения состояния этих индикаторов ЭВМ взаимодействует в обратном порядке с процессором клавиатуры, передавая ему специальные команды.

5.1.2. Конструкции клавиш

В современных клавиатурах используется несколько типов клавиш:

- с механическими переключателями;
- с замыкающими накладками;

- с резиновыми колпачками;
- мембранные.

В механических переключателях происходит замыкание металлических контактов. В них для создания "осязательной" обратной связи зачастую устанавливается дополнительная конструкция из пружины и смягчающей пластинки. При этом при нажатии ощущается сопротивление клавиш и слышится щелчок. Такие переключатели очень надежны, их контакты обычно самоочищающиеся. Они выдерживают до 20 миллионов срабатываний и стоят вторыми по долговечности после емкостных датчиков (см. ниже).

Клавиши с замыкающими накладками (см. рис. 22) широко применялись в старых клавиатурах, например фирмы Keytronics и др. В них прокладка из пористого материала с приклеенной снизу фольгой соединяется с кнопкой клавиши. При нажатии клавиши фольга замыкает печатные контакты на плате. Когда клавиша отпускается, пружина возвращает ее в исходное положение. Пористая прокладка смягчает удар при отпускании, но клавиатура при этом становится слишком "мягкой". Основной недостаток этой конструкции заключается в отсутствии щелчка при нажатии (нет обратной связи), поэтому в системах с такой клавиатурой часто приходится программным образом выводить на встроенный динамик компьютера какие-нибудь звуки, свидетельствующие о наличии контакта. Еще один недостаток такой конструкции состоит в том, что она весьма чувствительна к коррозии фольги и загрязнению контактов на печатной плате.

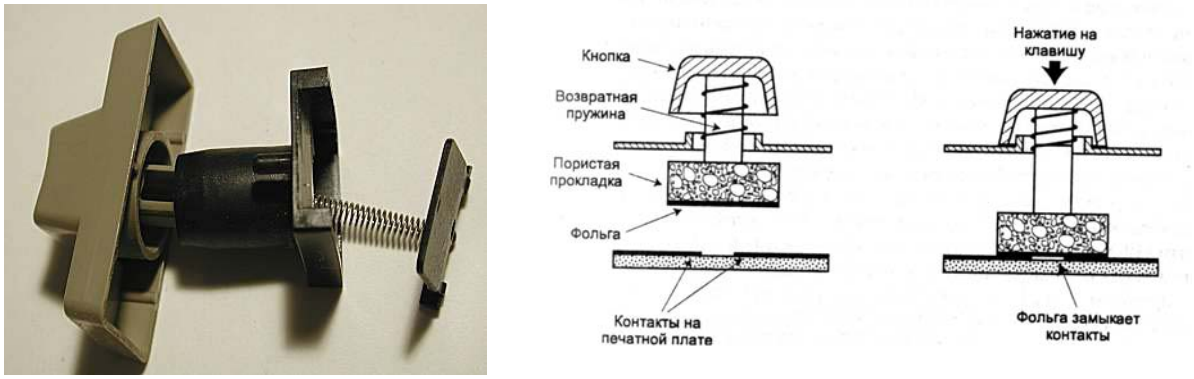


Рис. 22. Клавиши с замыкающимися накладками

Клавиатура с резиновыми колпачками (см. рис. 23) вместо пружины использует резиновый колпачок с замыкающей вставкой из той же резины, но с угольным наполнителем. При нажатии клавиши шток надавливает на резиновый колпачок, деформируя его. Деформация колпачка сначала происходит упруго, а затем он "проваливается". При этом угольный наполнитель замыкает проводники на печатной плате. При отпускании резиновый колпачок принимает свою первоначальную форму и возвращает клавишу в исходное состояние. Замыкающие вставки делаются из очищенного угля, потому они не подвержены коррозии и сами по себе очищают металлические контакты, к которым прижимаются. Колпачки обычно прессуются все вместе в виде листов резины, покрывающих плату целиком и защищающих ее от пыли, грязи и влаги.

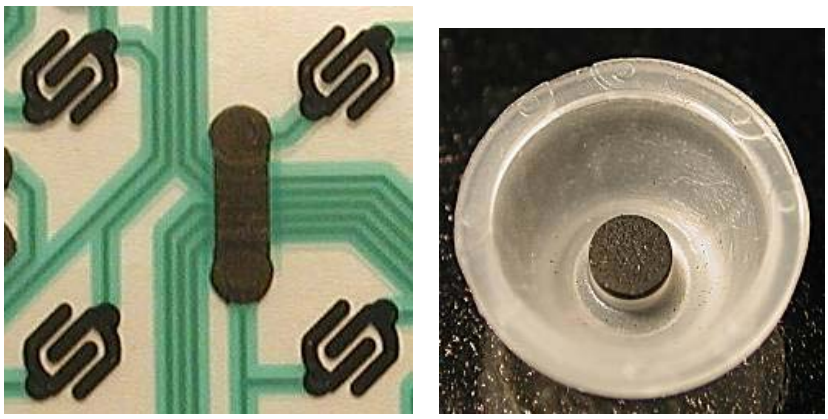


Рис. 23. Клавиши с резиновыми колпачками

Мембранная клавиатура является разновидностью предыдущей, но в ней нет отдельных клавиш: вместо них используется лист с разметкой, который укладывается на пластину с резиновыми колпачками. При этом ход каждой клавиши ограничен.

Емкостные датчики (см. рис. 24) являются единственными бесконтактными переключателями. В таких клавиатурах нет замыкающихся контактов. Их роль выполняют две смещающиеся относительно друг друга пластинки и специальная схема, реагирующая на изменение емкости между ними. Клавиатура представляет собой набор таких датчиков. При нажатии клавиши шток смещает верхнюю пластину ближе к неподвижной нижней пластине. Клавиши сконструированы так, что переход между пластинами происходит скачкообразно, и при этом слышен щелчок. Когда верхняя пластинка приближается к нижней, емкость между ними увеличивается, что регистрируется схемой компаратора, установленной в клавиатуре.

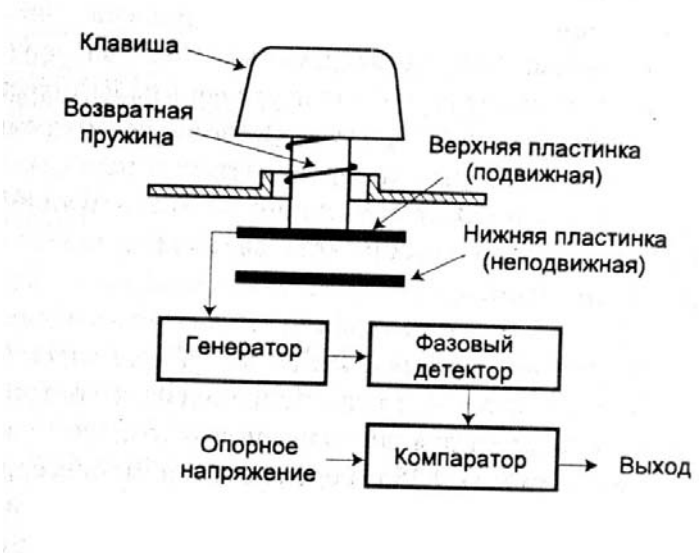


Рис. 24. Устройство емкостной клавиши

Из-за отсутствия электрических контактов такая клавиатура устойчива к коррозии и загрязнению. В ней практически отсутствует дребезжание (явление, когда при одном нажатии на клавишу символ вводится несколько раз подряд). Долговечность ее - до 25 миллионов срабатываний. Единственным недостатком

такой клавиатуры является ее высокая стоимость, но она во многом компенсируется удобством и долговечностью.

5.2. Монитор

Монитор, так же называемый дисплеем, - это устройство, предназначенное для отображения информации. По физическому принципу получения изображения мониторы можно разделить на: электронно-лучевые (ЭЛТ или CRT) и жидкокристаллические (LCD, TFT).

В первом случае картинка выводится с использованием стеклянной вакуумной электронно-лучевой трубки (см. рис. 24).

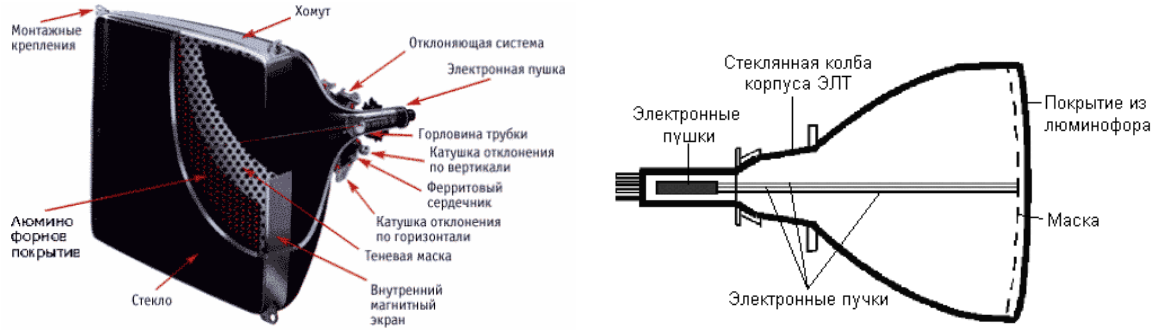


Рис. 25. Монитор с электронно-лучевой трубкой

С фронтальной стороны внутренняя часть стекла трубки покрыта специальным веществом – люминофором, испускающим свет при попадании на него заряженных частиц. В качестве люминофоров для цветных ЭЛТ используются довольно сложные составы на основе редкоземельных металлов - иттрия, эрбия и т.п.

Люминофор начинает светиться, под воздействием ускоренных электронов, которые создаются тремя электронными пушками, расположенными в противоположной стороне трубки. Каждая из трех пушек соответствует одному из основных цветов и посылает пучок электронов на различные люминофорные частицы, чье свечение основными цветами с различной интенсивностью комбинируется и в результате формируется изображение с требуемым цветом.

Наборы точек люминофора располагаются по треугольным триадам. Триада образует пиксел — точку, из которой формируется изображение (англ. pixel — picture element, элемент картинки).

Расстояние между центрами триад называется точечным шагом монитора. Это расстояние существенно влияет на чёткость изображения. Чем меньше шаг, тем выше чёткость. Обычно в цветных мониторах шаг составляет 0,28 мм и меньше. При таком шаге глаз человека воспринимает точки триады как одну точку "сложного" цвета.

Чтобы электроны беспрепятственно достигали экрана, из трубки откачивается воздух, а между пушками и экраном создаётся высокое электрическое напряжение, ускоряющее электроны. Перед экраном на пути электронов ставится маска — тонкая металлическая пластина с большим количеством отверстий, расположенных напротив точек люминофора. Маска обеспечивает попадание электронных лучей только в точки люминофора соответствующего цвета.

На ту часть колбы, где расположены электронные пушки, надевается отклоняющая система, заставляющая электронный пучок пробегать поочерёдно всю поверхность экрана. Количество отображённых строк в секунду называется **строчной частотой развертки**. А частота, с которой меняются кадры изображения, называется **кадровой частотой развёртки**.

Изображение на мониторах второго типа (жидкокристаллических) формируется при помощи специальных жидких кристаллов (Liquid Crystal) – органических веществ, способных под напряжением изменять величину пропускаемого света.

Жидкокристаллический монитор представляет собой две стеклянных или пластиковых пластины, между которыми находится суспензия (см. рис. 26). Кристаллы в этой суспензии расположены параллельно по отношению друг к другу, тем самым они позволяют свету проникать через панель. При подаче электрического тока расположение кристаллов изменяется, и они начинают препятствовать прохождению света.

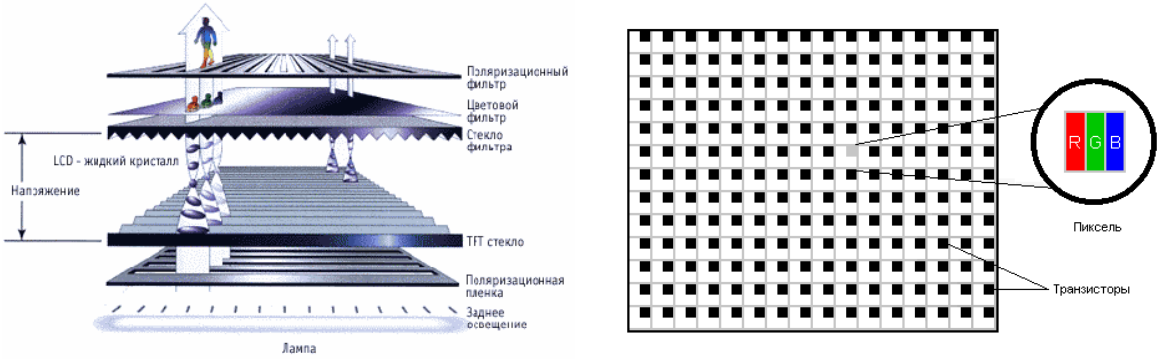


Рис. 26. Монитор с жидкокристаллическим экраном

Существует два вида жидкокристаллических мониторов: DSTN (dual-scan twisted nematic - кристаллические экраны с двойным сканированием) и TFT (thin film transistor - на тонкопленочных транзисторах), также их называют соответственно пассивными и активными матрицами. Такие мониторы состоят из следующих слоев: поляризующего фильтра, стеклянного слоя, электрода, слоя управления, жидких кристаллов, ещё одного слоя управления, электрода, слоя стекла и поляризующего фильтра.

Экран LCD-монитора представляет собой массив маленьких сегментов – пикселов. Как и в электроннолучевых трубках, пиксель формируется из трех участков - красного, зеленого и синего. Различные цвета получаются в результате изменения величины соответствующего электрического заряда (что приводит к повороту кристалла и изменению яркости проходящего светового потока).

5.3. Видеоадаптер

Для того чтобы монитор вывел картинку на экран, её надо сформировать. Для этого в персональном компьютере используется специальное устройство, называемое видеоадаптером (см. рис. 27).

Рис. 27. Вид видеоадаптера (ATI RADEON X850 XT PE)

Структура и состав видеоадаптера зависит от фирмы производителя, однако в общем виде это устройство состоит из следующих компонентов: памяти, микропроцессора, шинного интерфейса, ЦАП и ПЗУ.

Память служит непосредственно для хранения изображения, представляемого в виде набора точек – экранных пикселей. От объема памяти зависит максимально возможное разрешение видеокарты, определяемое как произведение трёх величин: количества столбцов, воспроизводимых на экране, количества строк и количества возможных цветов каждой точки. Таким образом, для разрешения 640x480x16 достаточно всего 256 кб. А для разрешения 1024x768x65536 уже требуется как минимум 2 Мб.

Многие современные мониторы используют для получения управляющих команд аналоговые сигналы, для формирования которых используется цифроаналоговый преобразователь (ЦАП или DAC).

Для начального запуска видеоадаптера и организации дальнейшего управления им используются специальные программы и данные, находящиеся в ПЗУ видеоадаптера.

Многие современные видеоадаптеры кроме функции генерации сигналов для монитора выполняют функции по обработке самого изображения. Зачастую это наложение нескольких изображений друг на друга. Например, курсор, текстуры и т.д. Для выполнения таких операций видеоадаптер оснащён собственным микропроцессором.

Шинный интерфейс предназначен для организации взаимодействия видеоадаптера с остальной частью ЭВМ. Наибольшее распространение получили интерфейсы типа AGP, PCI, PCI-X.

Для подключения современных мониторов используется два вида разъёмов (см. рис. 28): D-sub и DVI. Первый используется для подключения аналоговых мониторов (которые получают сигнал в аналоговой форме), второй - для подключения цифровых мониторов (получающие сигнал в цифровой форме).

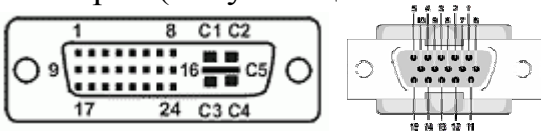


Рис. 28. Разъёмы для подключения мониторов (DVI – слева, D-sub – справа)

5.4. Терминалы – устройства ввода и вывода информации

Часто для взаимодействия с ЭВМ, в составе которых не предусмотрены собственные средства для взаимодействия с оператором (или они есть, но имеют скудный набор возможностей), используются специальные устройства называемые **терминалами**.

Чаще всего терминалы образуют единое устройство, соединенное с ЭВМ (или каким-то другим устройством) через кабельные или телефонные каналы (см. рис. 29). При этом к одной ЭВМ может подключаться несколько терминалов одновременно.

Примерами терминалов могут служить: POS-терминалы, операторские консоли по управлению промышленным оборудованием и т.п.

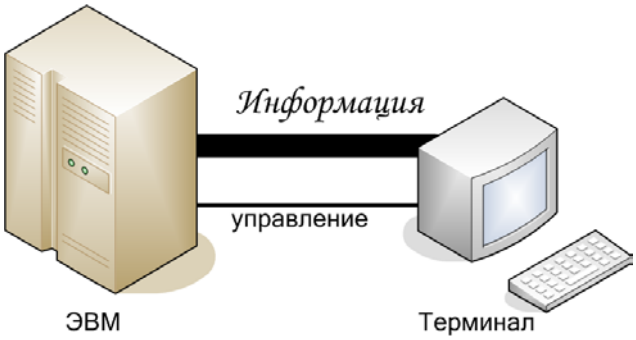


Рис. 29. Использование терминала для доступа к ЭВМ

Терминалы различаются возможностями устройств, входящих в их состав (т.е. сколько клавиш на клавиатуре, может ли монитор выводить графическую информацию или только текст, используется ли цвет для вывода информации на монитор и т.п.).

Первые терминалы были вроде дистанционно управляемых пишущих машинок, которые могли бы только "отображать" (печатать на бумаге) символьный поток, посланный им из компьютера. Самые ранние модели назывались «Телетайпами», они могли выполнять перевод строки и возврат каретки точно так же, как обыкновенная пишущая машинка. Такие терминалы стали называть пассивными, так как они только выводят информацию и не могут самостоятельно обрабатывать её.

Современные терминалы могут выполнять значительно больше действий, включающих ввод и вывод текстовой и графической информации, использование дополнительных каналов ввода информации, например, манипулятор «мышь», выполнять дополнительные функции, например распознавание штрих-кода и т.п. Состав и структура терминала определяется областью применения. Терминалы, способные проводить первичную обработку информации, называются активными. Далее будет рассматриваться именно такие терминалы.

Независимо от назначения и, соответственно, структуры терминала принцип его работы следующий (см. рис. 30).

Человек-оператор, нажимая клавиши на клавиатуре терминала, вводит информацию, которая поступает в устройство управления терминалом (УУТ). Далее она передается в ЭВМ в цифровом (численном) виде (скан-коды или

управляющие последовательности), и поступает специальной программе - драйверу, который её обрабатывает и передаёт выполняющейся программе (взаимодействующей с терминалом и ожидающей прихода от него данных). При необходимости, информация, поступающая от клавиатуры, может

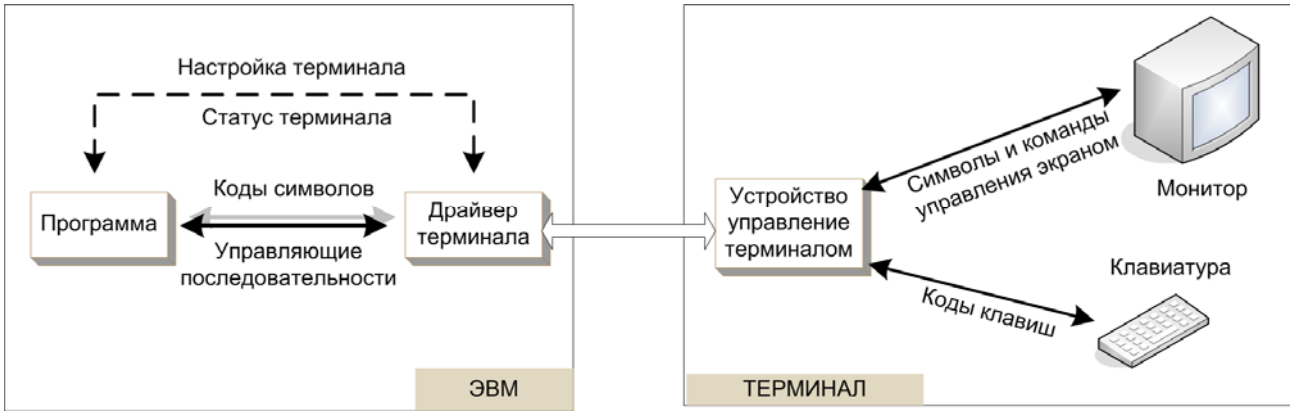


Рис. 30. Принцип работы терминала

сразу дублироваться на экране терминала. Такой режим называется «ЭХО».

Программа пользователя выводит результаты своей работы на терминал через драйвер, который передаёт её УУТ, а тот, в свою очередь, вырабатывает необходимые управляющие сигналы для монитора и выводит полученную информацию.

Ввод информации через терминал может осуществляться в одном из **двух режимов**: *каноническом*, в котором информация передается в ЭВМ только в виде законченных строк (т.е. после нажатия клавиши «ВВОД» или “ENTER”); и *неканоническом*, при котором вводимая информация сразу поступает в ЭВМ.

Помимо информационного потока между драйвером и УУТ передаются управляющие сообщения, которые позволяют настраивать УУТ на нужный режим работы и получать текущие его настройки. Программа пользователя также может управлять терминалом (например, передвинуть курсор на экране, или изменить цвет выводимых символов) путем передачи ему специальной последовательности символов, называемых **командами терминала** или **управляющими кодами**. Взаимодействовать напрямую с клавиатурой и монитором программа пользователя не может.

Формат и назначение команд терминала определяются его типом и (обычно) описываются производителем терминала в руководстве пользователя и в соответствующей программной среде (например, в ОС семейства UNIX (см. приложение) используется база данных настроек терминала, называемая terminfo).

Управляющие коды (или управляющие символы) обычно состоят из первых 32 байтов алфавита ASCII. Они включают такие коды: возврат каретки (переместить курсор к левому краю экрана), перевод строки (переместить курсор вниз на одну строку), возврат на один символ, символ ESC, табуляция и звонок. Они обычно не показываются на экране.

Так как не имеется достаточного количества управляющих кодов, чтобы делать все, используются команды терминала, чаше всего называемые **escape-последовательностями**. Они состоят из нескольких подряд идущих символов, первым из которых является символ с кодом ASCII 27, называемый "Escape" или ESC. Именно из-за первого символа команды терминала называются Escape-последовательностями.

После получения символа ESC, УУТ исследует символы после него так, чтобы интерпретировать их последовательность и выполнить соответствующую ей команду. Когда распознается конец последовательности, дальнейшие полученные символы отображаются на экране (если дальше опять не следует команда). Некоторые escape-последовательности могут иметь параметры (или аргументы), например, координаты на экране, куда надо переместить курсор. Параметры являются частью escape-последовательности.

Современные персональные компьютеры имеют клавиатуру и монитор непосредственно подключенные к их системному блоку. Функции по управлению этими устройствами возлагаются на операционную систему, которая воплощает в себе УУТ и драйвер терминала. Хотя производительность ПК достаточно высока и возможности по вводу и выводу информации практически неограниченны, они могут использоваться как терминалы для доступа к другим ПК или иным устройствам, к которым они подключены (например, видеосерверам, сетевым принтерам, модемам и т.п.). Для этого используются специальные программы, которые представляют ПК в виде псевдо или виртуального терминала. Примером таких программ можно назвать HyperTerminal, Remote Desktop, PuTTY, xterm и т.п. В некоторых операционных системах, например Linux, эмуляция терминала используется для того, чтобы позволить запустить несколько программ, выводящих различную информацию. При этом пользователь как будто работает за несколькими терминалами, поочередно переключаясь (например, нажимая комбинацию клавиш ALT+F1, или CTRL+ALT+F1), то на один, то на другой.

Используя графический режим вывода информации на монитор и программы эмуляции текстовых терминалов, пользователь может одновременно запустить несколько виртуальных терминалов.

5.4.1. Терминальные управляющие последовательности

Для описания всех доступных команд терминала и его возможностей в ОС Linux используется специальная база данных, называемая "termcap" (или "terminfo"). Она содержит разделы (отдельные файлы) для каждой модели терминала, в которых перечисляются все управляющие последовательности данного терминала и их синтаксис. Все разработчики терминалов должны придерживаться правил, указанных в этой базе.

Чтобы получить список доступных команд для определённого терминала можно использовать команду **infocmp** с параметрами –1L тип_терминала. Формат её вывода имеет вид:

поле = значение,

где «поле» - это название команды (например, clear_screen), «значение» - символьная последовательность, которую нужно отправить терминалу, чтобы выполнить указанную команду. Подробный перечень полей, которые могут быть описаны в базе termcap, можно найти в электронном справочнике man (man terminfo).

Символьная последовательность в поле «значение» может быть указана полностью, или приведены правила для её формирования. Для представления специальных символов используются записи вида \E (символ с кодом в ASCII 27 ESCAPE) или ^X (символ с кодом ASCII, рассчитываемом по формуле ‘X’ – ‘A’ + 1). Для указания правил используются форматные строки, аналогичные тем, что применяются в функции printf для указания формата вывода. Рассмотрим пример базы для терминала linux (текстового терминала).

Формат команды для получения содержимого базы следующий:

infocmp –1L linux.

Результат выполнения следующий (часть базы):

```
#     Reconstructed via infocmp from file: /usr/share/terminfo/l/linux
linux|linux console,
      acs_chars=+\020\054\021-
\030.^Y0\333`\004a\261f\370g\361h\260i\316j\331k\277l\332m\300n\305o~p\304q\304r
\304s_t\303u\264v\301w\302x\263y\363z\362{\343|\330}\234~\376,
      bell=^G,
      carriage_return=^M,
      clear_screen=\E[H\E[J,
      cursor_invisible=\E[?25l\E[?1c,
      cursor_visible=\E[?25h\E[?8c,
      enter_alt_charset_mode=\E[11m,
      enter_blink_mode=\E[5m,
      enter_bold_mode=\E[1m,
      exit_alt_charset_mode=\E[10m,
      key_f1=\E[[A,
      orig_colors=\E]R,
      restore_cursor=\E8,
      set_a_background=\E[4%p1%dm,
      set_a_foreground=\E[3%p1%dm
```

В результате получен список управляющих последовательностей, которые «понимает» терминал. Например, поле clear_screen (очистка экрана) содержит значение: \E[H\E[J. Это значит, что, если вывести на экран последовательность из 7 символов (\E, [, H, \E, [, 2, J), то произойдет очистка экрана и курсор переместится в левый верхний угол экрана. Интерес вызывает поле set_a_background (установить фон), значение которого равно: \E[4%p1%dm, что означает, что команда должна формироваться с использованием одного параметра (%p1), который должен иметь целый тип (%d) и располагаться между символами ‘4’ и ‘m’. Чаще всего текстовые терминалы поддерживают до 8 цветов, каждый из которых имеет свой номер. Например, если требуется установить цвет фона в белый (код 7), то команда должна иметь вид: \E[47m.

5.4.2. Основная и дополнительная таблица кодировок символов в терминалах

Существует множество различных типов терминалов, каждый из которых имеет свои особенности. Например, использует различные кодировочные таблицы, обрабатывает разный набор управляющих последовательностей и т.п. Одной из общих для всех текстовых терминалов проблем является правила вывода символов псевдографики (символов, позволяющих выводить текстовые рамки). Суть проблемы заключает в том, что расположение символов псевдографики в таблицах кодировок никак не регламентировано. Для того, чтобы позволить выводить символы псевдографики в любых типах терминалов (если они, конечно, имеют такую возможность) разработали универсальное правило, согласно которому в терминалах используется дополнительная кодировочная таблица, в которой располагаются требуемые символы. Чтобы переключить терминал на использование дополнительной таблицы, необходимо послеть ему управляющую команду, указанную в поле enter_alt_charser_mode. Команда, необходимая для обратного переключения, храниться в поле exit_alt_charset_mode.

Чтобы определить место расположения символов псевдографики, используют строку соответствия символов (поле acs_chars) тому расположению, который применяется в терминале типа VT100. В нём для вывода символом псевдографики используются символы из строки `afgijklmnopqrstuvwxyz{|}~. Значение каждого символа можно найти в электронном справочнике по слову terminfo.

5.5. Взаимодействие с терминалом в ОС Linux

Для взаимодействия пользователей с ПК, работающим под управлением операционной системы (ОС) Linux, используется эмуляция нескольких текстовых и графических терминалов. Т.е. другими словами, даже если пользователь сидит непосредственно за ПК, то он как будто работает за терминалом, подключенным к ЭВМ.

В ОС Linux пользователи "изолируются" от аппаратной части персонального компьютера. Для доступа к устройствам используется единый интерфейс в виде специальных файлов устройств, которые связывают приложения с соответствующими драйверами. Вся работа с устройством происходит через этот файл, а соответствующий ему драйвер обеспечивает выполнение операций ввода/вывода согласно конкретным протоколам обмена данными между ЭВМ и устройством. Причем правила работы с файлами устройств такие же, как и правила работы с файлами на запоминающем устройстве, с некоторыми дополнениями, позволяющими выдавать управляющие воздействия.

Все устройства, подключаемые к ЭВМ, условно можно разделить на два класса:

- блочные устройства, т.е. передающие и принимающие данные большими фрагментами, называемыми блоками или пакетами. При этом ядро операционной системы производит необходимую буферизацию. Примером

физических устройств, соответствующих этому типу файлов, являются жесткие диски.

- символьные устройства, т.е. использующие побайтную передачу данных. Терминалы относятся к таким устройствам.

Заметим, что это разделение условно, так как одно и тоже устройство может иметь и символьный и блочный интерфейс (т.е. уметь передавать данные, как побайтно, так и блоками).

Все специальные файлы устройств хранятся в каталоге /dev. Например, файл, соответствующий устройству «жесткий диск» имеет имя hda, а файл, соответствующий первому виртуальному терминалу – tty0.

Для взаимодействия со специальными файлами устройств используются функции прямого доступа к файлам - *open*, *read*, *write*, *close*. Для управления устройством используются системные вызовы *ioctl*. Для работы с терминалами используются дополнительные функции *isatty, tcgetattr, tcsetattr*, позволяющие произвести настройку драйвера на необходимый режим и определить текущее его состояние.

Обратите внимание (!!!), что все эти функции являются системными, в отличие от функций *fopen*, *fread*, *fwrite*, *fclose*.

5.5.1. Вызов open.

Системный вызов *open* используется для открытия файла. В качестве параметров в функцию передается строковая константа, соответствующая имени файла, который надо открыть, режим для открытия и дополнительные параметры.

Описание функций *open* и *close*

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open  (const char *pathname,  int flags);
int close (int fd);
```

В результате работы функции возвращается положительное целое число – номер дескриптора открытого файла, или -1, если во время открытия файла произошла ошибка. Дескриптор файла – это структура, описывающая файл. Хранится она в памяти пользовательской программы и может быть доступна по номеру дескриптора, который передается в качестве аргументов для функций, работающих с устройством.

Параметр *flags* определяет, в каком режиме требуется открыть файл. Он является целым числом, в котором за каждым битом однозначно устанавливается один требуемый режим. Для наглядности определения требуемых режимов в заголовочном файле *fcntl.h* библиотеки функций работы с файлами заданы соответствующие макросы. Например, режим открытия на чтение описывается макросом *O_RDONLY*, режим записи *O_WRONLY*, режим одновременной и записи и чтения (т.е. произвольного доступа) *O_RDWR*. Некоторые режимы могут

комбинироваться. Например, режим записи может комбинироваться с режимом дополнения файла, т.е. *flags* будет выглядеть как *O_APPEND | O_WRONLY*.

Обратной по действию функции *open* является функция *close*, которая закрывает указанный дескриптор.

Обратите внимание (!!!), что системные вызовы, в отличие от вызовов стандартной библиотеки Си, в качестве параметров принимают целое значение – номер дескриптора, а не указатель на структуру FILE.

При запуске каждой программы автоматически открываются три файла, соответствующие устройствам: ввода, вывода и вывода ошибок. В обычном случае все эти файлы соответствуют терминалу, с которого запущена программа. При необходимости пользователь может изменить эти устройства, например, перенаправив вывод программы в файл на диске или на принтер. Дескрипторы этих устройств имеют номера, соответственно, 0, 1 или 2. Поэтому все дескрипторы, создаваемые программами пользователей, нумеруются с цифры 3.

5.5.2. Вызовы isatty, ttyname

Для того чтобы проверить, является ли файл, описываемый некоторым дескриптором, специальным файлом терминала, используется вызов *isatty*. В качестве входного параметра эта функция принимает номер дескриптора, который надо проверить, и возвращает 1, если этот дескриптор связан с файлом терминала и 0 в противном случае.

Описание функции *isatty* и *ttyname*

```
#include <unistd.h>

int    isatty  (int desc);
char * ttyname (int fd);
```

Чтобы определить точное имя файла терминала, который был открыт под определённым номером, используется вызов *ttyname*. В качестве входного значения эта функция принимает номер дескриптора и возвращает указатель на строку, содержащую имя соответствующего файла, или NULL, если возникает ошибка (например, дескриптор не связан с файлом терминала).

Например, программа, проверяющая, какие потоки ввода, вывода и ошибок автоматически открыты для неё, представлена в листинге 1.

Листинг 1. Определение терминалов для потоков ввода, вывода и ошибок

```
1)  #include <stdio.h>
2)  #include <unistd.h>

3)  /* Основная функция программы */
4)  int main (void){
5)    /* Проверяем является ли дескриптор 0 файлом терминала */
6)    if (isatty(0)){
7)      printf ("Поток ввода связан с терминалом [%s]\n",
                  ttyname(0));
8)    } else {
9)      printf ("Поток ввода не связан с терминалом.\n");
```

```
10)    }
11)    /* Проверяем является ли дескриптор 1 файлом терминала */
12)    if (isatty(1)){
13)      printf ("Поток вывода связан с терминалом [%s]\n",
                  ttyname(1));
14)    } else {
15)      printf ("Поток вывода не связан с терминалом.\n");
16)    }
17)    /* Проверяем является ли дескриптор 2 файлом терминала */
18)    if (isatty(2)){
19)     printf ("Поток ошибок связан с терминалом [%s]\n",
                  ttyname(2));
20)    } else {
21)      printf ("Поток ошибок не связан с терминалом.\n");
22)    }
23)    return (0);
24)  }
25)
```

В строках 1-2 подключаются заголовочные файлы библиотек *stdio* и *unistd*, в которых описываются функции *printf*, *isatty*, *ttyname* используемые в программе.

Строка 4 определяет основную функцию программы. Её имя *main*. Она не принимает ни одного параметра и возвращает в операционную систему целое значение (результат работы программы).

В строках 6-10 проверяется, связан ли дескриптор стандартного потока ввода (он имеет номер 0) с файлом терминала (строка 6). Если он связан, то в стандартный поток вывода передается строка «Поток ввода связан с терминалом» и выводится полное имя этого файла (строка 7). В противном случае в поток вывода передается строка «Поток ввода не связан с терминалом» (строка 9).

В строках 11-22 аналогичным образом проверяется соответствие дескрипторов стандартных потоков вывода и вывода ошибок к подключению к терминалу.

Строка 23 завершает работу программы с результатом 0 (т.е. программа завершилась корректно).

5.5.3. Вызовы read, write

Чтение данных из устройства (точнее из его специального файла) производится с помощью функции *read*, которая в качестве параметров получает номер дескриптора, адрес буфера, куда необходимо поместить прочитанную информацию и максимальный размер этого буфера (т.е. может быть прочитано не более этого значения). Возвращает функций число прочитанных байтов или в случае ошибки -1.

Для передачи данных устройству (т.е. записи данных в специальный файл) используется вызов *write*. Параметры его аналогичны параметрам вызова *read* и возвращает он также число переданных байтов.

```
#include <unistd.h>

ssize_t read (int fd, void * buf, size_t count);
ssize_t write (int fd, void * buf, size_t count);
```

В качестве примера работы этих функций рассмотрим программу, считывающую последовательность данных с одного терминала и записывающая её на другой.

Обратите внимание (!!!), чтобы проверить, как работает эта программа, её необходимо запускать на одном из текстовых терминалов системы Linux. Для того, чтобы переключиться из графического терминала в текстовый, необходимо нажать комбинацию клавиш CTRL+ALT+F1. Чтобы вернуться в графический терминал, нужно нажать комбинацию клавиш ALT+F7. В ОС Linux имеется возможность работы с несколькими текстовыми и графическими терминалами (что и демонстрирует программа). Чтобы переключаться между текстовыми терминалами, используются комбинации клавиш ALT+F1, ALT+F2 и т.д. Соответственно ALT+F1 – переключает на первый терминал, ALT+F2 – на второй и т.д. Для демонстрации работы программы её необходимо запустить её на любом текстовом терминале, а на втором текстовом терминале зайти под своим учетным именем (так как программа попытается вывести информацию на 2-й терминал, а сделать она это сможет если 2-й терминал будет «принадлежать» Вам).

Листинг 2. Взаимодействия с терминалами

```
1)  #include <stdio.h>
2)  #include <sys/types.h>
3)  #include <sys/stat.h>
4)  #include <fcntl.h>
5)
6)  int main (void){
7)      int fd, read_chars;
8)      char buf[200];
9)
10)    /* Открываем файл для терминал 2 на запись */
11)     fd = open ("/dev/tty2", O_WRONLY);
12)     if (fd == -1){
13)       fprintf (stderr, "Ошибка открытия терминала.\n");
14)       return (1);
15)     }
16)
17)    /*  Читаем  с  клавиатуры  последовательность  символов  и
           выводим их на терминал 2 */
18)     if ((read_chars = read (0, buf, 199)) > 0){
19)        write (fd, buf, read_chars);
20)     } else {
21)        write (fd, "Ошибка", 6);
22)     }
23)     close (fd);
24) }
25)
```

5.5.4. Функции ioctl, tcgetattr, tcsetattr

Управление терминалом производится либо управляющими воздействиями, либо установкой значений атрибутов терминала. Первый способ осуществляется с использованием вызова *ioctl* (Input Output control), второй - вызовами *tcsetattr* и *tcgetattr*.

Описание функции *ioctl*

```
#include <sys/ioctl.h>

int ioctl (int fd, int request, ...);
```

Вызов *ioctl* в качестве входных значений принимает номер дескриптора открытого файла терминала, которым надо управлять, номер требуемой операции и дополнительные параметры, необходимые для выполнения этой операции. Вызов *ioctl* является универсальным и не зависит от типа устройства (т.е. он применяется для управления не только терминалами). Поэтому сама функция *ioctl* описана в заголовочном файле *sys/ioctl.h*, а номера функций, которые она может осуществлять над устройствами, в других заголовочных файлах. Функции взаимодействия с терминалами описаны в заголовочном файле termios.h. Примером управляющего воздействия является определение размера экрана терминала. Результат работы *ioctl* помещается в структуру *winsize*, которая имеет вид:

Описание структуры *winsize*

```
1)  struct winsize {
2)     unsigned short ws_row;
3)     unsigned short ws_col;
4)     unsigned short ws_xpixel;
5)     unsigned short ws_ypixel;
6)  }
```

Листинг 3. Определение размера экрана терминала

```
1)  #include <stdio.h>
2)  #include <termios.h>
3)  #include <sys/ioctl.h>
4)
5)  int main (void){
6)      struct winsize ws;
7)
8)      if (!ioctl(1, TIOCGWINSZ, &ws)){
9)        printf ("Получен размер экрана.\n");
10)       printf ("Число строк - %d\nЧисло столбцов - %d\n",
11)               ws.ws_row, ws.ws_col);
12)     } else {
13)       fprintf (stderr, "Ошибка получения размера экрана.\n");
14)     }
15)     return (0);
16) }
```

Параметры работы терминалов описываются структурой *termios*, состоящей из четырех регистров флагов, описывающих работу драйвера терминала при обработке входной информации (от клавиатуры), при выводе информации на экран, при передаче информации в ЭВМ, и дополнительных режимов, а также массив специальных управляющих символов:

Описание структуры *termios*

```
1)  struct termios{
2)     tcflag_t c_iflag;
3)     tcflag_t c_oflag;
4)     tcflag_t c_lflag;
5)     tcflag_t c_cflag;
6)     tcflag_t c_cc[NCCS];
7)  }
```

В зависимости от типа используемого терминала значение каждого из регистров флагов может изменяться. Здесь мы рассмотрим только наиболее важные из режимов работы текстового терминала ОС Linux. Более подробную информацию о настройке терминалов можно получить в электронном справочнике man по ключевому слову tty.

Регистр c_lflag описывает, как будет вести себя терминал при обработке и передаче информации в ЭВМ. Примером таких действий являются:

- определение режима работы (канонический или не канонический). Флаг называется ICANON;
- будут ли сразу отображаться на экране терминала вводимые с клавиатуры символы. Флаг называется ECHO;
- будут ли обрабатываться управляющие символы, например прерывание работы программы (CTRL+C) или приостановка работы программы (CTRL+Z). Флаг называется ISIG.

Если установлен флаг ICANON, то включается канонический режим работы терминала. Как уже было сказано выше, это позволяет использовать символы редактирования строки в процессе построчного ввода. Если флаг ICANON не установлен, то терминал находится в режиме прямого доступа (raw mode). Вызовы read будут при этом получать данные непосредственно из очереди ввода. Другими словами, основной единицей ввода будет одиночный символ, а не логическая строка. Программа при этом может считывать данные по одному символу или блоками фиксированного размера.

Если установлен флаг ISIG, то разрешается обработка клавиш прерывания (intr) и аварийного завершения (quit). Обычно это позволяет пользователю завершить выполнение программы. Если флаг ISIG не установлен, то проверка не выполняется, и символы intr и quit передаются программе без изменений. Значения управляющих символов задаются в массиве c_cc.

Если установлен флаг ECHO, то символы будут отображаться на экране по мере их набора. Сброс этого флага полезен для процедур проверки паролей и программ, которые используют клавиатуру для особых функций, например, для перемещения курсора или команд экранного редактора.

Массив c_cc содержит перечень символов, которые будут интерпретироваться как управляющие. Примером таких символов может служить символ с кодом 26 (комбинация клавиш CTRL+D), который интерпретируется как завершение ввода информации в каноническом режиме и т.д. Подробнее о назначении элементов массива c_cc можно прочитать в электронном справочнике man по ключевому слову tcgetattr.

Кроме этого, массив c_cc содержит ещё два элемента, описывающие поведение драйвера терминала при получении в не каноническом режиме запроса на чтение данных. А именно: какое количество символов должно быть в очереди, чтобы вызов *read* завершился (элемент, обозначаемый **VMIN**), и сколько времени (в десятых долях секунды) ждать появления хотя бы одного символа в очереди (параметр **VTIME**). Значения этих элементов определяются только для неканонического режима. В каноническом режиме они равны соответственно размеру буфера для строки и 0 (т.е. ожидать бесконечно долго).

Существуют четыре возможных комбинации параметров VMIN и VTIME:

- оба параметра VMIN и VTIME равны нулю. При этом возврат из вызова *read* обычно происходит немедленно. Если в очереди ввода терминала присутствуют символы (напомним, что попытка ввода может быть осуществлена в любой момент времени), то они будут помещены в буфер;
- параметр VMIN больше нуля, а параметр VTIME равен нулю. В этом случае *read* завершится только после того, как будут считаны VMIN символов. Это происходит далее в том случае, если вызов *read* запрашивал меньше, чем VMIN символов (т.е. ожидаться будут VMIN символом, а в буфер помещено требуемое число символов из полученных);
- параметр VMIN равен нулю, а параметр VTIME больше нуля. В этом случае вызов *read* завершится по приходу первого же символа или по истечению времени VTIME;
- оба параметра VMIN и VTIME больше нуля. В этом случае таймер запускается после получения первого символа, а не при входе в вызов *read*. Если VMIN символов будут получены до истечения заданного интервала времени, то происходит возврат из вызова *read*. Если таймер срабатывает раньше, то в программу пользователя возвращаются только символы, находящиеся при этом в очереди ввода.

Чтобы получить текущие настройки терминала используется вызов *tcgetattr*, который в качестве параметров получает номер дескриптора файла и адрес памяти, куда поместить структуру, описывающую режимы работы терминала. Результатом вызова будет либо 0, если параметры получены успешно, либо -1, если возникла какая-то ошибка.

```
#include <termios.h>

int tcgetattr (int fd, struct termios * tsaved);
int tcsetattr (int fd, int actions, struct termios * tnew);
```

Для установки новых параметров драйвера терминала используется вызов *tcsetattr*, которые в качестве параметров принимает номер дескриптора, новые значения флагов и правила их замены. Правила могут быть следующими:

- TCSANOW. Немедленное выполнение изменений, что может вызвать проблемы, если в момент изменения флагов драйвер терминала выполняет вывод на терминал;
- TCSADRAIN. Выполняет ту же функцию, что и TCSANOW, но перед установкой новых параметров ждет опустошения очереди вывода.
- TCSAFLUSH. Аналогично TCSADRAIN ждет, пока очередь вывода не опустеет, а затем также очищает и очередь ввода перед установкой для параметров дисциплины линии связи значений, заданных в структуре tnew.

ГЛАВА 6. ПОДСИСТЕМА ПРЕРЫВАНИЙ ЭВМ

Основной проблемой, возникшей в системах, основанных на принципе общей шины, стало простаивание процессора при взаимодействии с медленно работающими устройствами. Чтобы повысить эффективность использования процессора реализовали механизм его переключения на выполнение другой программы. Чтобы сообщить процессору, что медленное устройство снова готово взаимодействовать с ним, используется механизм прерываний.

Прерывание – это событие, происходящие в ЭВМ, при котором процессор временно приостанавливает выполнение одной (текущей) программы и переключается на выполнение другой программы, необходимой для обработки этого события. После окончания выполнения обработчика события, вызвавшего прерывание, процессор возобновляет выполнение приостановленной программы. Такой подход позволяет устройствам, входящим в состав ЭВМ, функционировать независимо от процессора, и сообщать последнему о своей готовности взаимодействовать с ним. Кроме этого, использование прерываний позволяет реагировать на особые состояния, возникающие при работе самого процессора (т.е. при выполнении им программ).

6.1. Механизм обработки прерываний

Механизм прерываний реализуется аппаратно-программными средствами. Для организации аппаратной системы прерываний используется контроллер прерываний, который подключается к соответствующему входу микропроцессора. К нему в свою очередь подключаются внешние устройства (см. рис. 31). Обычно контроллер может обрабатывать запросы от 8-ми источников, что на сегодняшний день является явно недостаточным. Поэтому используют каскадное подключение нескольких контроллеров, увеличивая тем самым число обслуживаемых внешних устройств. Чаще всего используют каскадное соединение только двух микросхем, обеспечивая 15 линий для генерации прерываний (одна используется для подключения ведомого контроллера). Программные прерывания реализуются самим микропроцессором.

Обработка прерываний (как аппаратных, так и программных) микропроцессором производится как минимум в четыре этапа:

- прекращение выполнения текущей программы;
- определение источника прерывания;
- выполнение обработчика прерывания;
- возврат к прерванной программе.

Первый этап должен обеспечить временное прекращение работы текущей программы таким образом, чтобы потом возможно было продолжить её выполнение. Любая программа, исполняемая процессором, располагается в своей области оперативной памяти и никак не затрагивает память других программ (если такие имеются). Разделяемым ресурсом (т.е. одновременно используемым) является сам процессор. Поэтому сохранить необходимо как минимум его состояние (точнее, состояние его внутренних регистров).

На втором этапе процессор определяет источник прерывания и сопоставляет ему адрес программы обработчика. Для этого используется специальная таблица прерываний, располагаемая в оперативной памяти. В этой таблице каждому источнику прерывания (а точнее его номеру) однозначно сопоставляется адрес оперативной памяти, где располагается программа обработчик.

После того, как определена программа обработчик, процессор начинает её исполнять, как будто это обычная программа (т.е. переключается). После её завершения автоматически загружается ранее прерванная программа и продолжается её выполнение.

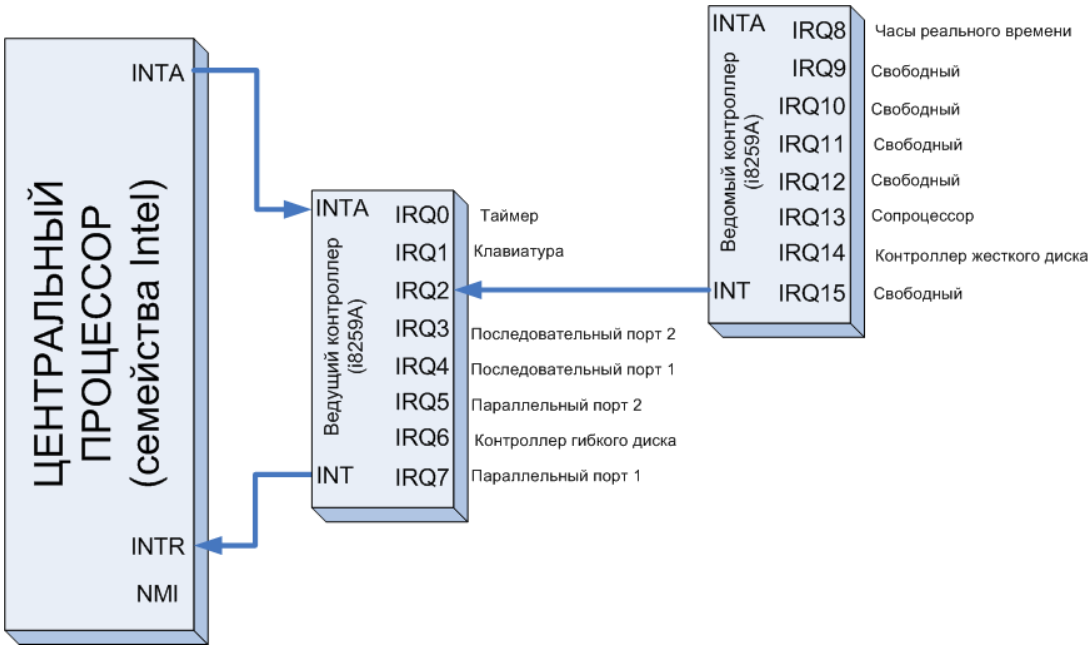


Рис. 31. Схема каскадного подключения контроллеров прерываний в ПК на базе процессоров семейства Intel

6.2. Обработка программных прерываний в UNIX системах. Сигналы

Аналогом программных прерываний в UNIX подобных операционных системах (например, Linux) служат сигналы. **Сигнал** – это способ взаимодействия программ, позволяющий сообщать о наступлении определённых событий, например появление в очереди управляющих символов или возникновение ошибки во время работы программы (например, Segmentation Faulit – выход за границы памяти).

Как и аппаратное прерывание, сигналы описываются номерами, которые описаны в заголовочном файле *signal.h*. Кроме цифрового кода, каждый сигнал имеет соответствующее символьное обозначение, например SIGINT.

Большинство типов сигналов предназначены для использования ядром операционной системы, хотя есть несколько сигналов, которые посылаются от процесса к процессу. Полный список доступных сигналов приведён в электронном справочнике man (man 7 signal). Приведем некоторые из них:

- SIGABRT - сигнал прерывания процесса (process abort signal). Посылается процессу при вызове им функции abort (). В результате сигнала SIGABRT произойдет аварийное завершение (abnormal termination) и запись образа памяти (core dump, иногда переводится

как <дамп памяти>). Образ памяти процесса сохраняется в файле на диске для изучения с помощью отладчика;

- SIGALRM - сигнал таймера (alarm clock). Посылается процессу ядром при срабатывании таймера. Каждый процесс может устанавливать не менее трех таймеров. Первый из них измеряет прошедшее реальное время. Этот таймер устанавливается самим процессом при помощи системного вызовов alarm или setitimer (см. ниже);
- SIGILL - недопустимая команда процессора (illegal instruction). Посылается операционной системой, если процесс пытается выполнить недопустимую машинную команду. Иногда этот сигнал может возникнуть из-за того, что программа каким-либо образом повредила свой код. В результате сигнала SIGILL происходит аварийное завершение программы;
- SIGINT - сигнал прерывания программы (interrupt). Посылается ядром всем процессам, связанным с терминалом, когда пользователь нажимает клавишу прерывания (т.е., другими словами в потоке ввода появляется управляющий символ, соответствующий клавише прерывания). Примером клавиши прерывания может служить комбинация CTRL+C. Это также обычный способ остановки выполняющейся программы;
- SIGKILL - сигнал уничтожения процесса (kill). Это довольно специфический сигнал, который посылается от одного процесса к другому и приводит к немедленному прекращению работы получающего сигнал процесса. Иногда он также посылается системой (например, при завершении работы системы). Сигнал SIGKILL - один из двух сигналов, которые не могут игнорироваться или перехватываться (то есть обрабатываться при помощи определенной пользователем процедуры);
- SIGPROF - сигнал профилирующего таймера (profiling time expired). Как было уже упомянуто для сигнала SIGALARM, любой процесс может установить не менее трех таймеров. Второй из этих таймеров может использоваться для измерения времени выполнения процесса в пользовательском и системном режимах. Сигнал SIGPROF генерируется, когда истекает время, установленное в этом таймере, и поэтому может быть использован средством профилирования (планирования работы) программы;
- SIGQUIT - сигнал о выходе (quit). Очень похожий на сигнал SIGINT, этот сигнал посылается ядром, когда пользователь нажимает клавишу выхода используемого терминала. Значение клавиши выхода по умолчанию соответствует символу ASCII F6 или Ctrl-Q. В отличие от SIGINT, этот сигнал приводит к аварийному завершению и сбросу образа памяти;
- SIGSEGV - обращение к некорректному адресу памяти (invalid memory reference). Сокращение SEGV в названии сигнала означает

нарушение границ сегментов памяти (segmentation violation). Сигнал генерируется, если процесс пытается обратиться к неверному адресу памяти. Получение сигнала SIGSEGV приводит к аварийному завершению процесса;

- SIGTERM - программный сигнал завершения (software termination signal). Используется для завершения процесса. Программист может использовать этот сигнал для того, чтобы дать процессу время для "наведения порядка", прежде чем посылать ему сигнал SIGKILL. Команда kill по умолчанию посылает именно этот сигнал;
- SIGWINCH – сигнал, генерируемый драйвером терминала при изменении размеров окна;
- SIGUSR1 и SIGUSR2 - пользовательские сигналы (user defined signals 1 and 2). Так же, как и сигнал SIGTERM, эти сигналы никогда не посылаются ядром и могут использоваться для любых целей по выбору пользователя.

При получении сигнала процесс может выполнить одно из трех действий:

- выполнить действие по умолчанию. Обычно действие по умолчанию заключается в прекращении выполнения процесса. Для некоторых сигналов, например, для сигналов SIGUSR1 и SIGUSR2, действие по умолчанию заключается в игнорировании сигнала. Для других сигналов, например, для сигнала SIGSTOP, действие по умолчанию заключается в остановке процесса;
- игнорировать сигнал и продолжать выполнение. В больших программах неожиданно возникающие сигналы могут привести к проблемам. Например, нет смысла позволять программе останавливаться в результате случайного нажатия на клавишу прерывания, в то время как она производит обновление важной базы данных;
- выполнить определенное пользователем действие. Программист может задать собственный обработчик сигнала. Например, выполнить при выходе из программы операции по «наведению порядка" (такие как удаление рабочих файлов), что бы ни являлось причиной этого выхода.

Чтобы определить действие, которое необходимо выполнить при получении сигнала, используется системный вызов **signal:**

Описание функции *signal*

```
#include <signal.h>

typedef void (*sighandler_t) (int);
sighandler_t signal (int signum,  sighandler_t handler);
```

Вызов *signal* определяет действие программы при поступлении сигнала с номером signum. Действие может быть задано как: адрес пользовательской функции (в таком случае в функцию в качестве параметра передается номер

полученного сигнала) или как макросы *SIG_IGN* (для игнорирования сигнала) и *SIG_DFL* (для использования обработчика по умолчанию).

Если действие определено как пользовательская функция, то при поступлении сигнала программа будет прервана и процессор начнет выполнять указанную функцию. После её завершения выполнение программы, получившей сигнал, будет продолжено и обработчик сигнала будет установлен как *SIG_DFL*.

Чтобы вызвать сигнал, используется системный вызов **raise**:

Описание функции *raise*

```
#include <signal.h>
int raise (int signum);
```

Процессу посылается сигнал, определенный параметром signum, и в случае успеха функция raise возвращает нулевое значение.

Чтобы приостановить выполнение программы до тех пор, пока не придет хотя бы один сигнал, используется вызов **pause**:

Описание функции *pause*

```
#include <signal.h>

int pause (void);
```

Вызов *pause* приостанавливает выполнение вызывающего процесса до получения любого сигнала. Если сигнал вызывает нормальное завершение процесса или игнорируется процессом, то в результате вызова *pause* будет просто выполнено соответствующее действие (завершение работы или игнорирование сигнала). Если же сигнал перехватывается, то после завершения соответствующего обработчика прерывания вызов *pause* вернет значение -1 и поместит в переменную errno значение EINTR.

Пример программы, обрабатывающей прерывания приведён в листинге 1.

Листинг 1. Обработка сигналов

```
1)  #include <stdio.h>
2)  #include <signal.h>
3)
4)  /* Функция-обработчик сигнала */
5)  void sighandler (int signo){
6)    printf ("Получен сигнал !!! Ура !!!\n");
7)  }
8)  /* Основная функция программы */
9)  int main (void){
10)   int x = 0;
11)
12)   /* Регистрируем обработчик сигнала */
13)   signal (SGUSR1, sghandler);
14)   do {
15)     printf ("Введите X = "); scanf ("%d", &x);
16)     if (x & 0x0A) raise (SIGUSR1);
```

```
17)    } while (x != 99);
18) }
```

6.3. Работа с таймером

Как было сказано ранее, каждая программа в UNIX-подобных операционных системах может устанавливать три таймера:

- **ITIMER_REAL** уменьшается постоянно и подает сигнал SIGALRM, когда значение таймера становится равным 0;
- **ITIMER_VIRTUAL** уменьшается только во время работы процесса и подает сигнал SIGVTALRM, когда значение таймера становится равным 0.
- **ITIMER_PROF** уменьшается во время работы процесса и когда система выполняет что-либо по заданию процесса. Совместно с ITIMER_VIRTUAL этот таймер обычно используется для профилирования времени работы приложения в пользовательской области и в области ядра. Когда значение таймера становится равным 0, подается сигнал SIGPROF.

Когда на одном из таймеров заканчивается время, процессу посылается сигнал и таймер обычно перезапускается.

Для установки таймеров используется функция *setitimer*. Величина, на которую устанавливается таймер, определяется следующими структурами:

Описание структур *itimerval* и *timeval*.

```
struct itimerval {
   struct timeval it_interval; /* следующее значение */
   struct timeval it_value; /* текущее значение */
};

struct timeval {
   long tv_sec;  /* секунды */
   long tv_usec; /* микросекунды */
};
```

Значение таймера уменьшается от величины it_value до нуля, после чего генерируется соответствующий сигнал, и значение таймера вновь устанавливается равным it_interval. Таймер, установленный на ноль (его величина it_value равна нулю или время вышло, и величина it_interval равна нулю), останавливается. Величины tv_sec и tv_usec являются основными при установке таймера.

Если устанавливаемое время срабатывания таймера измеряется в полных секундах, то для установки таймера может использоваться системный вызов alarm:

Описание функций alarm и setitimer

```
#include <unistd.h>

unsigned int alarm(unsigned int secs);

#include <sys/time.h>

int setitimer(int which, struct itimerval *value, struct itimerval
*ovalue);
```

Функция *alarm* запускает таймер, который через secs секунд сгенерирует сигнал SIGALRM. Поэтому вызов *alarm(60)*; приводит к посылке сигнала SIGALRM через 60 секунд. Обратите внимание, что вызов *alarm* не приостанавливает выполнение процесса, как вызов *sleep*, вместо этого сразу же происходит возврат из вызова *alarm*, и продолжается нормальное выполнение программы, по крайней мере, до тех пор, пока не будет получен сигнал SIGALRM. «Выключить» таймер можно при помощи вызова *alarm* с нулевым параметром: *alarm(0)*.

Вызовы *alarm* не накапливаются. Другими словами, если вызвать *alarm* дважды, то второй вызов отменит предыдущий. Но при этом возвращаемое вызовом *alarm* значение будет равно времени, оставшемуся до срабатывания предыдущего таймера.

Пример программы, настраивающей терминал и обрабатывающей сигнал от него приведен в листинге 2.

Листинг 2. Использование таймера

```
1)  #include <stdio.h>
2)  #include <signal.h>
3)  #include <sys/time.h>
4)
5)  void signalhandler (int signo){
6)      printf ("Сработал таймер\n");
7)  }
8)
9)  int main (void)
10) {
11)   struct itimerval nval, oval;
12)
13)   signal (SIGALRM,  signalhandler);
14)
15)   nval.it_interval.tv_sec = 3;
16)   nval.it_interval.tv_usec = 500;
17)   nval.it_value.tv_sec = 1;
18)   nval.it_value.tv_usec = 0;
19)
20)   /* Запускаем таймер */
21)   setitimer (ITIMER_REAL, &nval, &oval);
22)
23)   while (1){
24)     pause();
25)   }
26)
27)   return (0);
28) }
```

ГЛАВА 7. НАКОПИТЕЛИ НА ЖЕСТКИХ МАГНИТНЫХ ДИСКАХ

Практически во всех современных персональных компьютерах для долговременного хранения информации применяются устройства, использующие магнитные или оптические свойства различных материалов.

Устройствами хранения данных, использующими магнитный принцип, являются дисководы гибких дисков, жесткие диски, стримеры (устройства хранения данных на магнитных лентах), ZIP, АРВИД и т.п. Здесь внимание будет уделено основным принципам работы жестких дисков, так как в современных персональных компьютерах они наиболее часто используются для хранения информации.

Накопители на жестких дисках часто называют «винчестерами». Так их стали называть после того, как фирма IBM в начале 1970-х годов выпустила первое подобное устройство. Это был громоздкий 14-дюймовый диск, который имел обозначение "30/30", так как позволял записать 30 дорожек по 30 секторов в каждой из них. Обозначение диска напоминало название широко распространенной модели ружья фирмы "Winchester", в результате чего для обозначения дисковых устройств с несъемными дисками стали широко применять это слово.

7.1. Конструкция дисководов жестких дисков

Конструктивно жесткий диск (см. рис. 32) - это корпус, содержащий тонкие металлические диски, покрытые магнитным слоем, над которыми перемещаются головки чтения/записи, смонтированные на шпинделе.

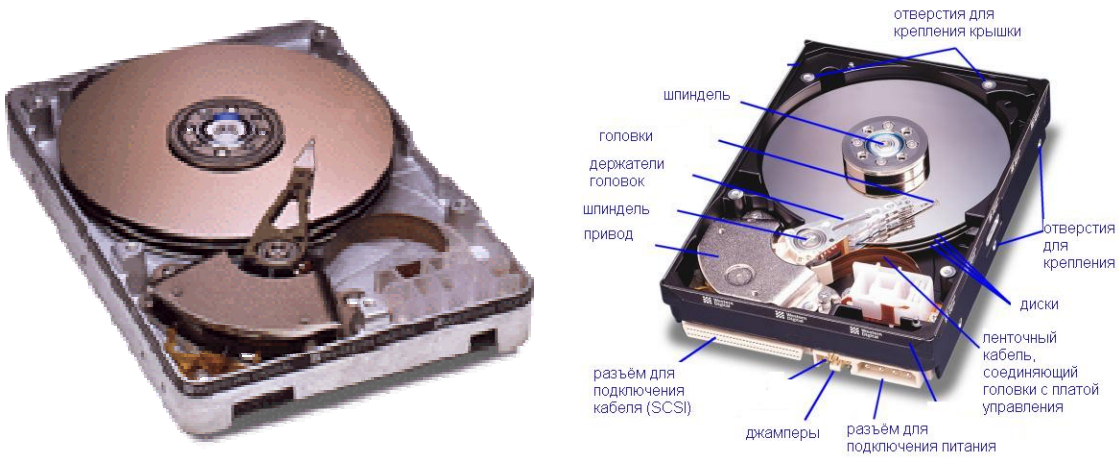


Рис. 32. Устройство хранения данных на жестких магнитных дисках (винчестер)

Магнитные диски – это пластины из алюминия, стекла или керамики с нанесенным на них слоем высококачественного ферромагнетика. Такие диски, в отличие от дискет (или гибких дисков), довольно сложно согнуть. Именно поэтому они называются «жесткими».

Независимо от того, какой материал используется в качестве основы диска, он покрывается тонким слоем вещества, способного сохранять остаточную намагниченность после воздействия внешнего магнитного поля. Самыми распространенными являются два типа слоя:

- *оксидный* – полимерное покрытие с наполнителем из оксида железа;

- *тонкоплёночный* – состоящий из нескольких слоев с различными материалами, рабочим из которых является слой из сплава кобальта толщиной около 0,025 мкм.

Покрытия на основе окислов железа и бариевых ферритов являются достаточно мягкими, поэтому их использование в новых разработках почти прекратилось. Металлические пленочные покрытия обеспечивают более высокую плотность записи и прочность поверхности диска, которая особенно важна при использовании дисков в переносных компьютерах, где велика вероятность ударов.

Стеклянная основа для дисков делает возможным получение поверхности, менее критичной к температуре и позволяющей наносить информацию с более высокой плотностью. Кроме того, в перспективе планируется применять в качестве основы для магнитных дисков полимерную (пластиковую основу), чтобы ещё больше увеличить плотность записи данных.

Магнитные диски собирают в один пакет и закрепляют на оси, устанавливаемой в привод, который вращает их со скоростью до 15000 об/мин. Обычно в пакете содержится от 2 до 11 дисков. Очевидно, что чем быстрее вращается диск, тем быстрее можно найти на нем требуемое место, но при очень большой скорости вращения пластины могут разрушиться.

Чтобы получить доступ к информации, используются **магнитные головки** чтения/записи, являющиеся электромагнитными элементами. В процессе записи головка преобразует электрический сигнал в электромагнитный импульс, способный изменить магнитные свойства покрытия диска в конкретном заранее установленном месте. Во время чтения информации контроллеру жесткого диска головка перемещается на нужное место и начинает "снимать" состояние магнитного слоя и преобразовывать его в электрический сигнал, который декодируется в двоичную последовательность данных.

Головка всегда находится на некотором расстоянии от поверхности диска (около 0.13 мкм), обеспечиваемом за счет потока воздуха при быстром вращении диска (головка "летит"). Уменьшение зазора между головкой и поверхностью диска увеличивает сигнал при считывании и позволяет снизить ток записи, однако сильно снижает устойчивость устройства к вибрациям и ударам. Тем не менее, работы по уменьшению зазора между диском и головкой не прекращаются ведущими производителями винчестеров.

Наличие зазора между головкой и поверхностью диска требует **парковки головок** (перемещения их за пределы рабочей поверхности) при выключении компьютера во избежание повреждения поверхности диска или головки при их механическом контакте. В старых устройствах для парковки головок нужно было использовать специальные программы (их запускали непосредственно перед выключением компьютера), современные винчестеры при выключении питания перемещают головки за пределы рабочей зоны дисков автоматически.

При изготовлении головок используются три различных технологических варианта:

- монолитные головки - изготавливаются из ферритов. Сложность обработки и хрупкость ферритов накладывают серьезные ограничения на их ис-

пользование в современных системах с высокой плотностью записи информации на диск. В новых разработках такие головки почти не используются;

- композитные головки - имеют меньшие размеры по сравнению с монолитными и выполнены из феррита на подложке из стекла или твердой керамики. Такой подход позволяет уменьшить зазор между головкой и поверхностью диска и, как следствие, повысить плотность записи на диск. Некоторые фирмы при производстве композитных головок используют вместо воздушного зазора в магнитном сердечнике головки зазор, заполненный металлом (это позволяет улучшить конфигурацию магнитного поля головки и дополнительно увеличить плотность записи);
- тонкопленочные головки создаются методом фотолитографии. Магнитный сердечник головки осаждается на керамическую поверхность, что позволяет создать головки с очень малым магнитным зазором. Такая технология дает самую высокую плотность записи и позволяет уменьшить ширину дорожек.

Головки чтения/записи крепятся на специальном приводе, который управляет позицией головок относительно поверхности диска. От типа используемого привода непосредственно зависит скорость работы устройства в целом - привод обеспечивает один из основных параметров винчестера: *время позиционирования головок* (seek time). Для перемещения головок обычно используются шаговые двигатели, обеспечивающие высокую точность позиционирования.

Существуют два различных варианта приводов перемещения головок: линейные и поворотные. При поворотном приводе головки перемещаются по дуге окружности, линейный привод обеспечивает перемещение головок по радиусу диска. Преимущество линейного привода заключается в том, что зазор магнитной головки всегда перпендикулярен дорожке и расстояние между дорожками постоянно. Поворотные приводы обеспечивают меньшую инерционность и, как следствие, более быстрое позиционирование головок. Кроме того, поворотные приводы более устойчивы к ударам и вибрации, поскольку допускают точную балансировку.

В самом первом магнитном накопителе, разработанном фирмой IBM, диски и головки вместе с несущей конструкцией размещались в отдельном закрытом корпусе (его называли модулем данных), устанавливаемом для работы на приводное устройство. При установке модуля данных на привод автоматически подключалась система подачи в модуль данных очищенного воздуха. Головки, благодаря малой массе, прижимались к поверхности диска с усилием всего 0.1Н, а при вращении диска между головкой и поверхностью образовывался воздушный зазор толщиной около 0.5мкм.

В современных устройствах модуль данных и привод составляют единое целое. Система подачи очищенного воздуха уже не используется. Для надежной и качественной работы винчестера важно обеспечить отсутствие пыли в корпусе блока дисков и головок, для чего широко используются барометрические фильтры, выравнивающие давление внутри и снаружи блока дисков.

На каждом жестком диске кроме блока дисков и привода установлена печатная плата - контроллер управления (как правило, он крепится снизу), обеспечивающая управление приводами головок и дисков, а также усиление сигналов записи/считывания. Кроме того, на этой плате установлен дешифратор команд управления головками, схемы стабилизации и др. На современных винчестерах, изготавливаемых в рамках программы Energy Star, имеется также устройство, обеспечивающее отключение привода дисков при отсутствии запросов к устройству и другие функции энергосбережения.

Жесткий диск имеет несколько разъёмов:

- интерфейсный разъём (для подключения к ЭВМ);
- разъём питания;
- иногда разъем для заземления корпуса диска.

Форм-фактор интерфейсного разъема определяет стандарт взаимодействия жесткого диска с контроллером на материнской плате. Наибольшее распространение получили стандарты IDE (или ATA) и SCSI. В обычных персональных компьютерах используются диски с интерфейсом ATA.

7.2. Адресация данных на жестких дисках

Вся информация на диске записывается в форме концентрических окружностей, называемых **дорожками** (см. рисунок 33). Расстояние между дорожками определяется шагом перемещения головок чтения/записи, которые располагаются над каждой поверхностью диска. Совокупность дорожек, расположенных друг над другом, называется **цилиндром**. Каждая дорожка поделена на дуги – секторы, которые являются минимальной единицей информации на жестком диске.

Сектор состоит из трех основных частей: заголовка, тела (512 байт), заключения. Информация в заголовке определяет начало и номер сектора. В заключении хранится контрольная сумма, необходимая для проверки целостности данных.

Для того, чтобы контроллер жесткого диска считал или записал какой-то сектор, необходимо ему передать номер соответствующих головки, цилиндра и сектора. Такая адресация называется **CHS** (от англ. Cylinder/Head/Sector). Принято секторы нумеровать целыми числами, начиная с единицы, а цилиндры и головки – целыми числами, начиная с нуля.

Геометрией диска называется совокупность характеристик, позволяющих определить максимальный объем хранимой информацией. Другими словами, геометрия - это максимальное число цилиндров (C), число головок (H) и число секторов на дорожку (S).

Емкость = HxCxSx(размер сектора). Например, если имеется винчестер с 4-мя дисками, каждый из которых поделён на 100 дорожек по 200 секторов (по 512 байт) каждая, то его ёмкость будет равняться – (4*2)x100x200x512 = 81920000 байт (или 80 000 Кбайт). Необходимо помнить, что на один диск приходится по 2 головки и один Кбайт равен 1024 байт.

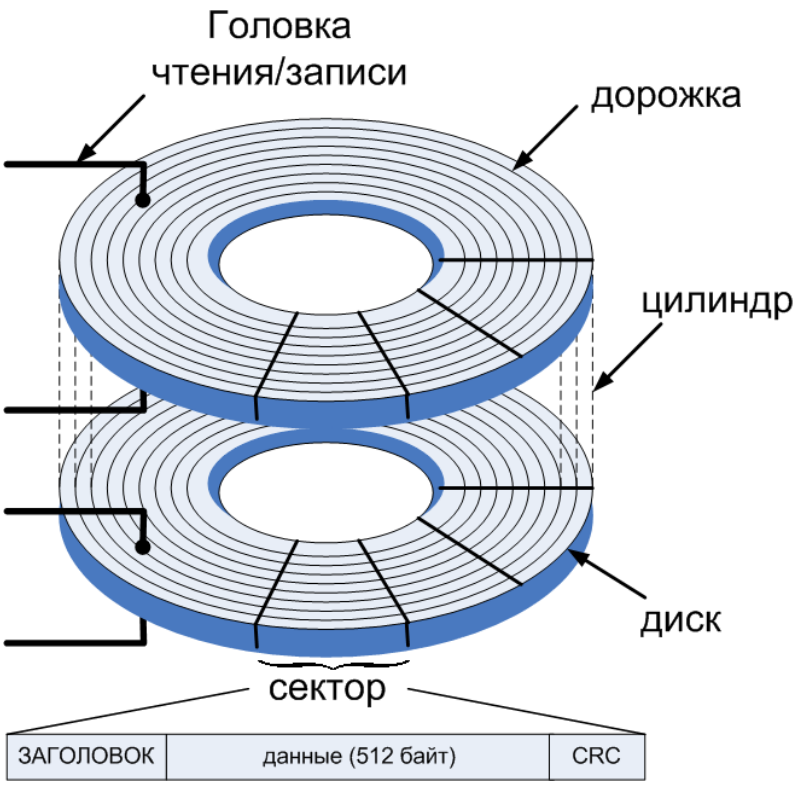


Рис. 33. Способ хранения информации на жестком диске

Для передачи адреса сектора используются три двоичных числа с определённой разрядностью. Изначально стандарт ATA определял для указания номера цилиндра 16 бит, для номера сектора - один байт (8 бит), для номера головки 4 бита. Такая адресация дисков имела ограничение в 128 ГБайт при размере сектора в 512 байт.

Со временем, когда размера диска в 128 Гбайт стало не хватать, стали применять **линейную** адресацию секторов, в которой все сектора на диске нумеровались целыми числами, начиная с 0. Получая такой адрес, контроллер жесткого диска по определённому правилу преобразовывает его в номера соответствующих головок, цилиндров и секторов.

При использовании линейной адресации секторов геометрией диска является всего лишь максимально возможный номер сектора.

Для того чтобы сохранить принцип преемственности программного обеспечения (т.е. чтобы на новой аппаратуре могли работать старые программы) используется трансляция (см. рисунок 34) CHS-адресации и геометрии в LBA и наоборот. Это преобразование осуществляется специальной программой – драйвером, который используется для доступа программ пользователя к информации на жестком диске.

Чаще всего, в качестве такого драйвера используется одна из функций BIOS. Однако некоторые операционные системы используют собственные драйверы и взаимодействуют напрямую с жестким диском.

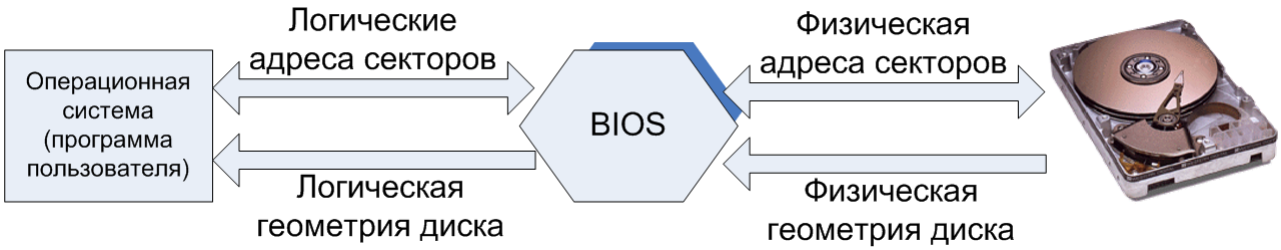


Рис. 34. Принцип трансляции адресов секторов и геометрий жестких дисков

7.3. Барьеры размеров жестких дисков. Трансляция адресов

Стремясь сократить объем памяти, требуемый для адресации секторов на диске, в старых версиях BIOS были приняты ограничения для адресации секторов на диске. В них для "C" отводилось 10 бит, для "H" отводилось 8 бит, для "S" отводилось 6 бит. Для хранения CHS адреса используется три 8-разрядных ячейки: первая - H, вторая - 6 младших разрядов соответствуют номеру сектора, 2 старших – 2-м старшим разрядам номера цилиндра, третья ячейка – 8-мь младшим разрядам номера цилиндра.

Таким образом, получилось **ограничение BIOS** (или DOS, которая использовала только драйвер BIOS) **в 8.5 ГБ**. Сегодня это является серьезным ограничением на размер диска. А именно, DOS не в состоянии использовать большие диски, так как она использует только драйвер BIOS.

При использовании таких ограничений на величины C,H,S был получен очередной «барьер размера жесткого диска». **Ограничения ATA и BIOS** складываются так, что никто не может использовать больше чем 1024 цилиндра, 16 считывающих головок и 63 сектора, что составляет 528482304 байт (**504 МБ**).

Для того, чтобы преодолеть барьер в 512 Мбайт, первоначально стали использовать прием (сейчас в современных BIOS он называется Large) уменьшения числа цилиндров (кратно степени 2) с одновременным увеличением числа головок. Таким образом, появилась возможность использовать диски объемом до 8,4 ГБайт, но с "фальшивой" (логической) геометрией. При чтении информации с жесткого диска BIOS переводит адрес из одной формы в другую. Именно поэтому такой способ называется **логической адресацией** секторов на диске. Число, на которое было уменьшено количество цилиндров и увеличено количество головок, используется в дальнейшем при преобразовании логического адреса сектора в физический адрес.

Очевидно, что при любой трансляции геометрии жесткого диска будет происходить потеря размера диска (за счет округлений значений чисел до целых).

Например, пусть имеет винчестер со следующей физической геометрией - C = 1238, H = 16, S = 63 (объем = 638 Мбайт), логическая адресация будет следующей - C = 619, H = 32, S = 63 (объем = 638 Мбайт). Теперь все программное обеспечение, использующее функцию BIOS для доступа к диску, считает, что работает с жестким диском, в котором 32 головки вместо 16 и 619 цилиндров вместо 1238.

Перевод из логического CHS-адреса в физический CHS (и обратно) осуществляется следующим образом (см. рис. 35):

$$\textit{прямое} \qquad S_ф = S_л, \qquad C_ф = C_л^Г \left(H_л \% K\right) + C_л, \quad H_ф = \left[H_л / K\right]$$

$$\textit{обратное} \quad S_л = S_ф, \qquad H_л = \frac{H_л^Г}{K}\left(C_ф \% K\right) + H_ф, C_л = C_ф \% C_л^Г$$

где $С_л^Г$ - число цилиндров в логической геометрии. $Н_ф^Г$ - число головок в физической геометрии. K - делитель, полученные при преобразовании физической геометрии в логическую геометрию. Запись $\left[x\right]$ - означает целочисленное деление. Запись x%y – означает получение остатка от целочисленного деления x на y.

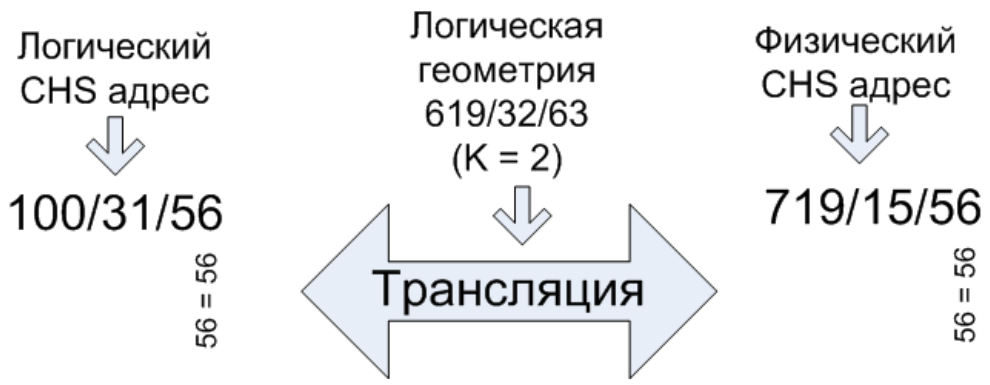


Рис. 35. Преобразование CHS адресов при использовании Large трансляции

Для получения логической CHS геометрии при работе с дисками LBA драйвер BIOS так же применяет механизм трансляции.

В этом случае для получения геометрии BIOS выполняет следующие действия. Считается, что на диске все дорожки всегда делятся на 63 сектора. Номер максимального сектора делится на 63 (логическое число секторов). Полученное число (L1) делится на 1023 и округляется до ближайшего (большего) числа из ряда: 2, 4, 8, 16, 32, 64, 128, 255. Полученное число равно числу головок в логической геометрии (Нлог). Остаток от деленния L1 на Нлог будет число цилиндров в логической геометрии (Слог).

Например, имеется жесткий диск с геометрией LBA 10018890. Делим его на 63. Получаем 159030. Делим его на 1023. Получаем 155. Округляем до 255. Затем 159030 делим на 255 и получаем 623. Таким образом, логическая C/H/S геометрия принимает вид - 623/255/63 (т.е. считается, что имеется 623 цилиндра (0-622), 255 головки(0-254), 63 сектора (1-63)).

Преобразование адресов в режиме LBA осуществляется следующим образом (см. рис. 36):

$$\textit{прямое} \qquad S_{\text{LBA}} = \left(C_л \cdot H_л^Г + H_л\right) \cdot \text{S}_л^Г + \text{S}_л - 1,$$

$$\text{обратное} \quad \text{S}_л = \left(S_{LBA} \% S_л^Г\right) + 1, \quad H_л = \left(\left[S_{\text{LBA}} / S_л^Г\right]\right) \% H_л^Г, \quad C_л = \left[\left[S_{\text{LBA}} / S_л^Г\right] / H_л^Г\right]$$

где S_{LBA} - номер максимально возможного сектора в LBA геометрии. $S_л^Г$ - число секторов в логической геометрии.

Рис. 36. Трансляция адреса из CHS формата в LBA

В современных винчестерах для экономии поверхности дисков делают дорожки с разным количеством секторов. Другими словами, дорожки с большим радиусом делят на большее число секторов. Несмотря на это, контроллер жесткого диска взаимодействует с BIOS стандартным образом, и сам преобразует получаемые адреса.

7.4. Логическая организация винчестера. Таблица разделов

В современных персональных компьютерах жесткие диски логически разделяются на одну или несколько областей, называемых разделами или томами. Это позволяет одновременно использовать несколько операционных систем на одном компьютере.

Информация о том, на какие разделы поделён жесткий диск, хранится в виде **таблиц разделов**, располагаемых в специально отведённых секторах. Обычно для формирования этих таблиц используются служебные программы, например FDISK.

При подготовке любого жесткого диска в его первый сектор (цилиндр 0, головка 0, сектор 1) заносится информация, называемая **главной загрузочной записью** (MBR - Master Boot Record). В ней содержится **программа загрузчик ядра операционной системы** (446 байт), основная **таблица разделов** (64 байт), с которой начинается поиск информации и разбиение жесткого диска на разделы и два байта 55h и 44h (используются для проверки правильности заполнения этого сектора).

Назначение и содержание программы загрузчика было рассмотрено ранее.

Таблица разделов содержит 4 записи по 16 байт, описывающих части диска. Структура записи приведена в табл. 3. Обратим внимание, что в ней присутствует как CHS, так и LBA адресация для того, чтобы как старые операционные системы (работающие только по CHS), так и новые могли определить разбиение диска на разделы. При этом CHS адресация может описать разделы, начало и конец которых располагаются в области, не превышающей размер в 8 Гбайт.

Первым байтом в структуре, описывающей раздел, идет флаг активности раздела (0 - не активен, 128 (80H) - активен). Он служит для определения, явля-

ется ли раздел системным загрузочным и необходимо производить загрузку операционной системы с него при старте компьютера. Активным может быть только один раздел.

Далее следует 24-разрядный CHS адрес сектора, с которого начинается раздел. Формат его записи следующий: 1 байт – соответствует номеру головки (H), биты 0-5 второго байта соответствуют номеру сектора (S), биты 6-7 второго байта – битам 8-9 номера цилиндра (C), третий байт соответствует битам 0-7 номера цилиндра (C). Такая форма записи применяется для того, чтобы поместить CHS адрес в три 8-битовые ячейки.

Таблица 3. Формат таблицы разделов

Название записи элемента Partition Table	Длина байт
Флаг активности раздела	1
CHS адрес начального сектора раздела	3
Кодовый идентификатор операционной системы	1
CHS адрес конечного сектора раздела	3
LBA адрес начального сектора	4
Размер раздела в секторах	4

Затем следует байт, определяющий тип раздела, и описывающий, содержит ли этот раздел данные или он является дополнительным (или расширенным), т.е. содержащим другие разделы, которые не описаны в текущей таблице. Те разделы, которые не описаны в основной таблице (т.е. в той, которая записана в первом секторе диска) называются логическими. Значение кодов приведены в таблице 4.

За байтом кода операционной системы следует CHS адрес сектора, которым завершается раздел (формат аналогичен формату описания начального сектора).

Далее идет адрес начального в формате LBA и размер раздела в секторах.

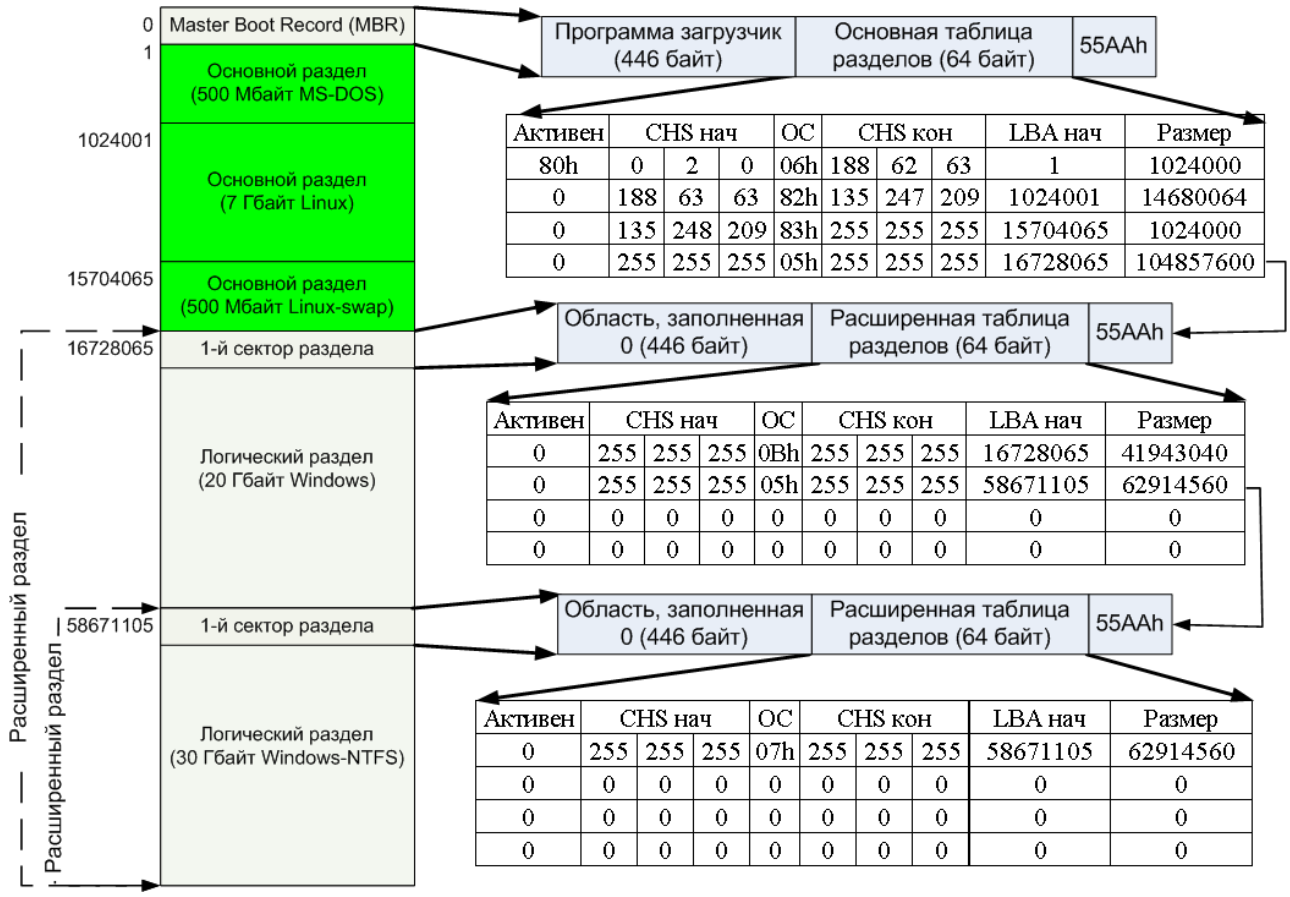
Таблица 4. Типы разделов жесткого диска

Код	Значение	Код	Значение
0x00	Empty	0x07	HPFS/NTFS
0x01	FAT12	0x0c	Win95 FAT32 (LBA)
0x04	FAT16 <32M	0Eh	Win95 FAT16
0x05	Расширенный	0x82	Linux swap
0x06	MS-DOS FAT16	0x83	Linux

Описание логических разделов является цепочкой таблиц, расположенной внутри расширенного раздела (см. рис. 37). Структура таблиц, описывающих логические разделы, такая же, как и в главной таблице разделов. Запись, имеющая идентификатор системы, указывает на сектор диска, где находится

начальная таблица описания таблицы логических дисков. Кроме этого, в таблице может быть запись, имеющая также код расширенного раздела, и указывающая смещение относительно текущего.

Следует помнить, что операционные системы семейства Microsoft каждому разделу сопоставляют букву латинского алфавита, начиная с буквы «C:». При этом основные разделы именуются первыми, а только затем логические. Если встречаются разделы других операционных систем, то они не именуются и, соответственно, получить доступ к ним становится невозможным.



Активен	CHS нач			OC	CHS кон			LBA нач	Размер
80h	0	2	0	06h	188	62	63	1	1024000
0	188	63	63	82h	135	247	209	1024001	14680064
0	135	248	209	83h	255	255	255	15704065	1024000
0	255	255	255	05h	255	255	255	16728065	104857600

Активен	CHS нач			OC	CHS кон			LBA нач	Размер
0	255	255	255	0Bh	255	255	255	16728065	41943040
0	255	255	255	05h	255	255	255	58671105	62914560
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Активен	CHS нач			OC	CHS кон			LBA нач	Размер
0	255	255	255	07h	255	255	255	58671105	62914560
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Рис. 37. Пример полной таблицы разделов для жесткого диска размером 60 ГБайт

ГЛАВА 8. БАЗОВЫЕ ЭЛЕМЕНТЫ ЭВМ. АЗЫ БУЛЕВОЙ АЛГЕБРЫ

Основными элементами, на которых строятся ЭВМ, являются простые цифровые устройства - вентили, которые могут принимать два значения – «ноль» и «единица». Чаще всего состоянию «единица» соответствует положительное напряжение (например, +5В) на его выходе, состоянию «ноль» - нулевое напряжение. Схемы из вентилей могут вычислять различные функции от этих двух значений. Эти элементы формируют основу для построения сложных цифровых схем.

8.1. Базовые элементы для построения ЭВМ

В основу работы вентиля положен принцип работы транзисторов, которые, в определённых условиях, могут работать как бинарные переключатели. На рисунке 38а изображен биполярный транзистор, имеющий три контакта: *коллектор, эмиттер* и *базу*. Если входное напряжение V_{in} на базе ниже критического значения, т.е. соответствует логическому нулю, то транзистор закрывается и выступает как сопротивление. При этом напряжение V_{out} на выходе высокое, т.е. соответствует логической единице. Если V_{in} превышает критическое значение, т.е. равно логической единице, то транзистор открывается и V_{out} стремится к нулю. Такая схема называется **инвертером**, т.е. преобразовывает входное значение на противоположное. Для переключения транзистора из одного состояния в другое требуется некоторое время, например, несколько наносекунд.

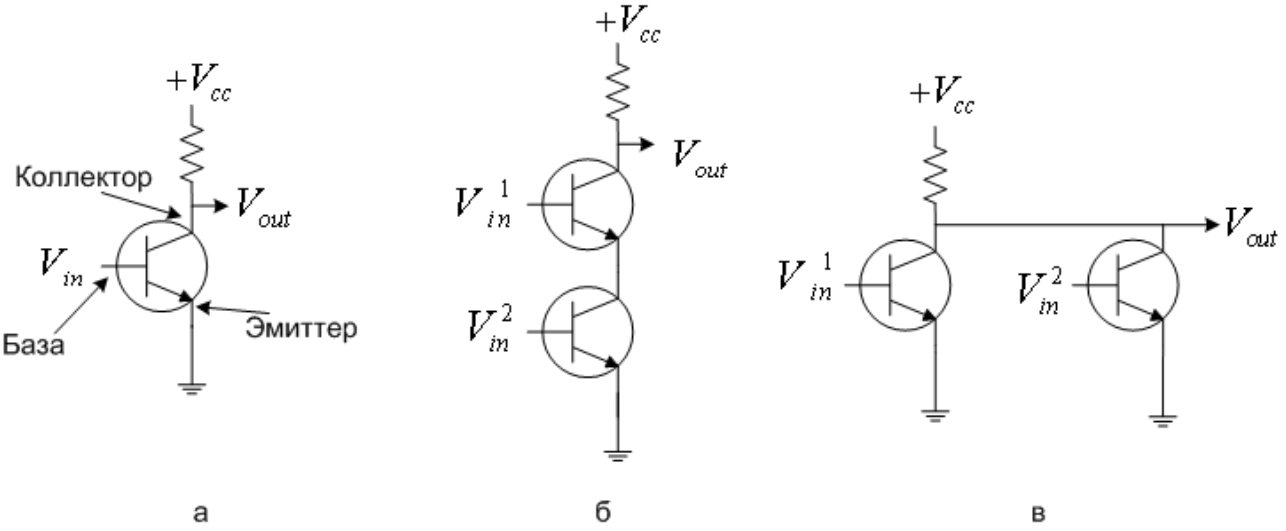


Рис. 38. Схемы вентилей НЕ (а), И-НЕ (б), ИЛИ-НЕ (в)

Если два транзистора соединить последовательно (см. рис. 38б), то схема начинает выполнять функции логического сложения с инверсией. В этом случае напряжение V_{out} будет высоким только в том случае, если закрыты оба транзистора, т.е. напряжения V_{in}^1 и V_{in}^2 высокие, т.е. соответствуют логическим единицам.

Если два транзистора соединить параллельно (см. рис. 38в), то схема начинает выполнять функции логического умножения с инверсией. Если напряжения V_{in}^1 и V_{in}^2 высокие, то оба транзистора открыты и напряжение V_{out} стремится к 0. Если же хотя бы одно их напряжений V_{in}^1 или V_{in}^2 низкое, то соответствующий транзистор находится в закрытом состоянии и напряжение V_{out} отлично от 0, т.е. соответствует логической единице.

Эти три схемы образуют три простейших вентиля, и называются НЕ, И-НЕ, ИЛИ-НЕ соответственно. Если выход схем И-НЕ и ИЛИ-НЕ подключить на вход схеме НЕ, то получатся схемы И и ИЛИ. В первой схеме на выходе будет единица только в том случае, если на оба входа схемы подается единица. На выходе второй схемы получится единица, если хотя бы на один из входов подается единица.

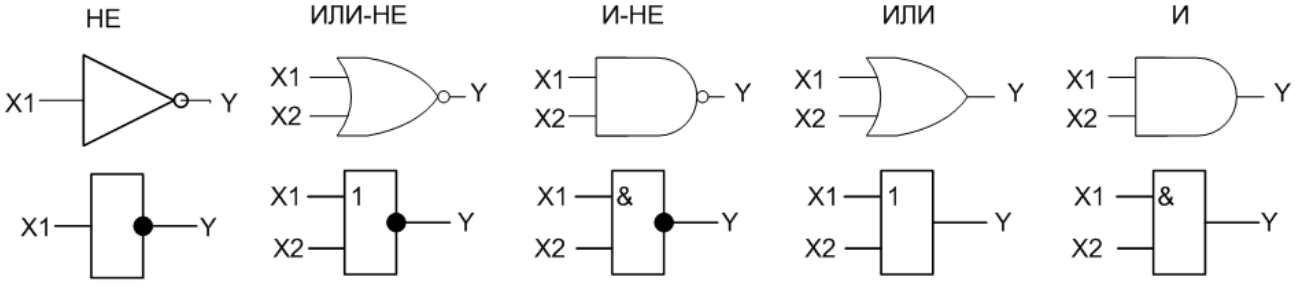
Из выше сказанного ясно, что для реализации схем И-НЕ и ИЛИ-НЕ требуется всего два транзистора, а для схем И и ИЛИ – три. По этой причине чаще всего используются только вентили НЕ, И-НЕ и ИЛИ-НЕ.

8.2. Элементы булевой алгебры

Чтобы математически описать схемы, которые строятся путём сочетания различных вентилей, используется особый тип алгебры, называемой **булевой алгеброй**, в которой все переменные и функции могут принимать только значения 0 или 1. Название эта алгебра получила в честь английского математика Джорджа Буля.

Как и в обычной алгебре, правила преобразования входных значений в выходные называются *функциями*, которые могут зависеть от одной или несколько переменных, и получать результат (0 или 1), основываясь только на их значениях. Если функция содержит n переменных, то существует 2^n возможных наборов значений функции.

Булева функция может быть представлена двумя способами: в виде таб-



X1	Y
0	1
1	0

X1	X2	Y
0	0	1
0	1	0
1	0	0
1	1	0

X1	X2	Y
0	0	1
0	1	1
1	0	1
1	1	0

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Рис. 39. Значки для изображения вентилей на схемах и булевы функции, описывающие их работу (сверху вниз – европейское обозначение, российское обозначение, булева функция в виде таблицы истинности)

лицы истинности и с помощью алгебраической записи.

Если написать таблицу, в которой перечислить всевозможные комбинации входных переменных и соответствующие этим комбинациям значения функций (выходное значение), то получится **таблица истинности**. Очевидно, что размер таблицы определяется числом переменных в функции и может быть огромным (2^n, где n - число входных переменных).

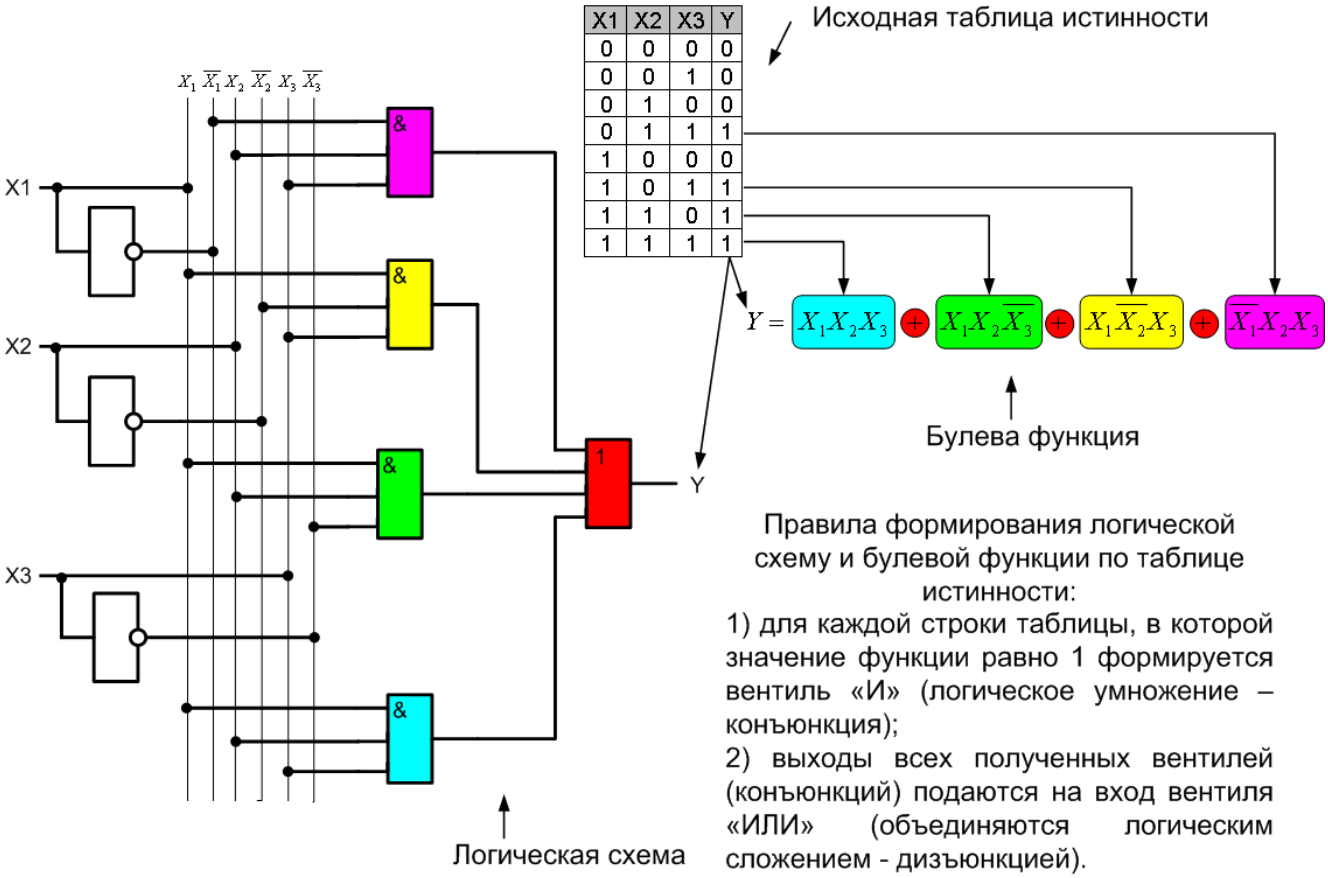


Рис. 40. Формирование булевой функции и синтез логических схем

Как альтернатива таблицы истинности используется **алгебраическая запись**, в которой перечисляются все комбинации переменных, дающие единичное (или нулевое) значение функции. При этом знаком «умножить» обозначается операция И, а знаком плюс – операция ИЛИ, черта над переменной означает операцию НЕ. Например, для таблицы истинности вида:

X_1	X_2	X_3	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

функция примет вид: $Y = \overline{X_1}\,\overline{X_2}\,\overline{X_3} + \overline{X_1}\,\overline{X_2}X_3 + X_1\overline{X_2}\,\overline{X_3} + X_1\overline{X_2}X_3$. Здесь первое слагаемое образуется из инверсных значений входных переменных, так как

единица на выходе соответствует их нулевым значениям, второе слагаемое – из двух инверсных и одного прямого, так как единица на выходе соответствует нулевым значениям двух переменных и единичному значению третьей и т.д.

8.3. Простейший синтез цифровых схем

Схематически вентили НЕ, И-НЕ, ИЛИ-НЕ, И, ИЛИ обозначаются, как показано на рис. 39. Для отличия вентилей И-НЕ и ИЛИ-НЕ от И и НЕ на схемах первых выход обозначается кругом.

Синтез логической схемы осуществляется образом, аналогичным получению булевой функции из таблицы истинности (см. рис. 40). Для каждой строки таблицы истинности (или для каждого слагаемого булевой функции, если схема строится на основе неё) формируется вентиль «И». Все полученные вентили объединяются через вентиль «ИЛИ», в результате чего получается выход схемы.

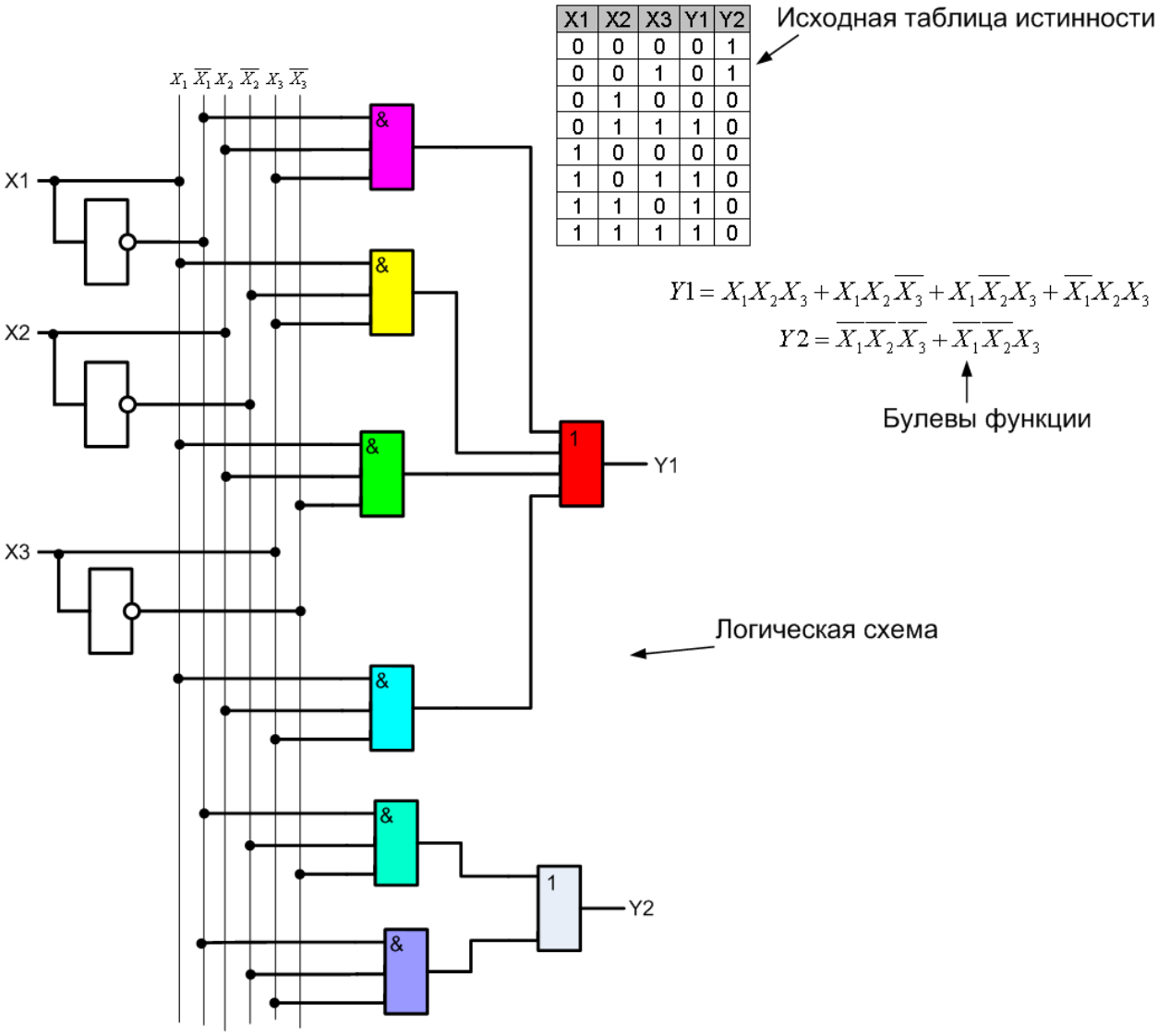


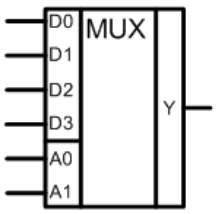
Рис. 41. Синтез схемы для устройства, имеющего два выхода

Конечно, представленный подход не гарантирует получения оптимальных схем (имеющих минимальное число вентилей и построенных на основе минимального количества типов вентилей), однако позволяет построить схему

для любой булевой функции. Вопросы организации оптимальных схем и минимизации булевых функций выходят за рамки данной лабораторной работы и поэтому рассматриваться не будут.

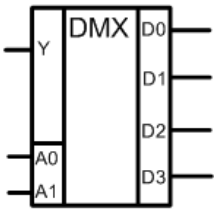
Мультиплексор

D0	D1	D2	D3	A0	A1	Y
0	*	*	*	0	0	0
1	*	*	*	0	0	1
*	0	*	*	0	1	0
*	1	*	*	0	1	1
*	*	0	*	1	0	0
*	*	1	*	1	0	1
*	*	*	0	1	1	0
*	*	*	1	1	1	1



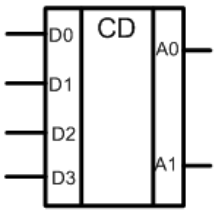
Демультиплексор

Y	A0	A1	D0	D1	D2	D3
0	0	0	0	0	0	0
1	0	0	1	0	0	0
0	0	1	0	0	0	0
1	0	1	0	1	0	0
0	1	0	0	0	0	0
1	1	0	0	0	1	0
0	1	1	0	0	0	0
1	1	1	0	0	0	1



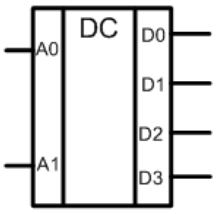
Шифратор

D0	D1	D2	D3	A0	A1
1	0	0	0	0	0
*	1	0	0	0	1
*	*	1	0	1	0
*	*	*	1	1	1



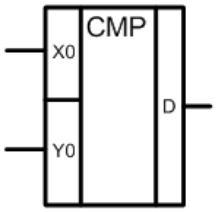
Дешифратор

A0	A1	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



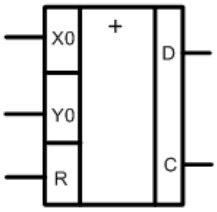
Компаратор

X0	Y0	D
0	0	1
0	1	0
1	0	0
1	1	1



Сумматор

R	X0	Y0	C	D
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Сдвиговый регистр

X0	X1	C	Y0	Y1
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

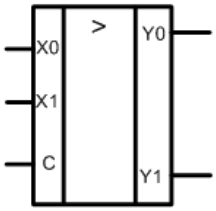


Рис. 42. Комбинаторные схемы, их таблицы истинности и обозначение на логических схемах (символ * в таблице истинности означает любое значение)

8.3.1. Комбинационные цифровые логические схемы

Многие применения цифровой логики требуют схемы с несколькими входными и несколькими выходными сигналами, которые получаются только исходя из значений входных сигналов. Такие схемы называются **комбинационными.** В качестве примеров можно назвать:

- мультиплексоры - устройства с 2^n информационными входами, одним выходом и n линиями управления, которые выбирают один из входов системы и соединяют его с выходом. Другими словами, мультиплексор – это цифровая схема канала для поступающей информации. Схема, обратная мультиплексору (т.е. имеющая один вход, соединяемый с несколькими выходами) называется демультиплексором;
- шифраторы - устройства, имеющие 2^n входов и n выходов, на которых выдается двоичное число, соответствующее номеру входа с единичным значением. Схема, обратная шифратору (т.е. сопоставляющая n-разрядное число выходу с соответствующим номером), называется дешифратором. Обычно, если единица подаётся на несколько входов шифратора, то во внимание берётся только та, которая располагается в старшем разряде;
- компараторы – устройства, имеющие $2n$ входов (т.е. входов для поступления двух n-разрядных чисел) и один выход, на котором устанавливается единичное значение, если входные числа равны (т.е. у них совпадают все разряды);
- схемы, выполняющие арифметические действия:
  - регистры сдвига - устройства, имеющие n информационных и один управляющий вход и n выходов, на которых формируется n-разрядное число, равное входному n-разрядному числу сдвинутому влево, если на управляющий вход подаётся единица, и сдвинутому вправо, если на управляющий вход подаётся ноль. Кроме информационных выходов, имеется дополнительный выход, сигнализирующий о **переполнении**, т.е. ситуации, в которой результат операции выходит за n разрядов;
  - сумматоры - устройства, имеющие $2n$ входов и n выходов, на которых формируется результат арифметического сложения. Кроме эstого, предусматривается выход, сигнализирующий о переполнении и вход, учитывающий перенос с предыдущего разряда (R).

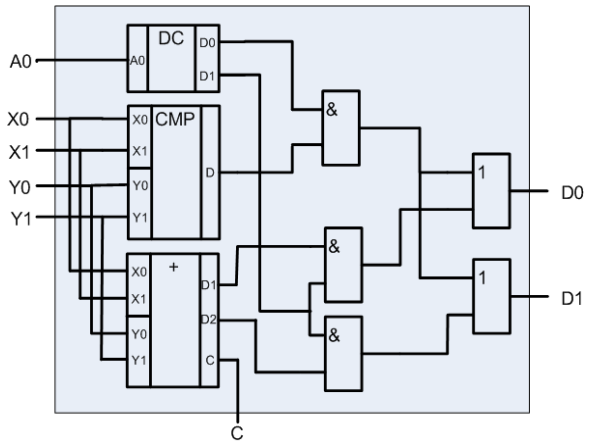


Рис. 43. Пример схемы АЛУ, выполняющего два действия

Таблицы истинности для рассмотренных комбинационных схем приведены на рис. 42.

В ЭВМ схемы подобного рода объединяются в одно функциональное устройство, называемое **арифметико-логическим устройством** (АЛУ). Все операции, которые может выполнять АЛУ, имеют свой уникальный номер, называемый **кодом операции**. В зависимости от того, какой код операции подается на вход АЛУ, будет выбрана соответствующая часть схемы (используя дешифратор и схему И) и выполнено требуемое действие. Например, на рис. 43 изображено АЛУ, выполняющее два действия – сложение и сравнение двух чисел.

СПИСОК ЛИТЕРАТУРЫ

1. В. Юров. Ассемблер: Учебник – СПб.: Питер, 2002. – 624 с.
2. Кутузов М.А., Преображенский А. Выбор и модернизация компьютера. – СПб.: Питер, 2004. – 320 С.
3. Скотт Мюллер, Крег Зекер. Модернизация и ремонт ПК, 10-е изд.: Пер. с англ. – К.; М.; СПб.: Издательский дом «Вильямс», 1999 .- 992 С.
4. Томпсон Р. Б., Томпсон Б. Ф. Железо ПК: Энциклопедия. 3-е изд. - СПб.: Питер, 2004. – 960 С.
5. Хамахер К., Вранешич З., Заки С. Организация ЭВМ. 5-е изд. - СПб.: Питер, 2003. – 848 С.
6. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: Учебник для вузов. – СПб.: Питер, 2004. – 668 С.
7. Э. Танненбаум. Архитектура компьютера: 4-е изд. – СПб.: Питер, 2003. – 704 С.

ПРИЛОЖЕНИЕ 1. ТАБЛИЦА СКАН-КОДОВ

КОДЫ КЛАВИШ ДЛЯ 101-,102- и 104-х КЛАВИШНЫХ КЛАВИАТУР ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

КЛАВ.	КОД	ОТЖАТИЕ		КЛАВ.	КОД	ОТЖАТИЕ		КЛАВ.	КОД	ОТЖАТИЕ
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	FO,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	FO,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	FO,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	05	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	06	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	04	F0,04		KP 4	6B	F0,6B
X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	03	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	01	F0,01		]	5B	F0,5B
3	26	F0,26		F10	09	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78		'	52	F0,52
5	2E	F0,2E		F12	07	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-				

КОДЫ КЛАВИШ УПРАВЛЕНИЯ ПИТАНИЕМ

КЛАВ.	КОД	ОТЖАТИЕ
Power	E0, 37	E0, F0, 37
Sleep	E0, 3F	E0, F0, 3F
Wake	E0, 5E	E0, F0, 5E

КОДЫ КЛАВИШ РАСШИРЕНИЯ WINDOWS

КЛАВ.	КОД	ОТЖАТИЕ
Next Track	E0, 4D	E0, F0, 4D
Previous Track	E0, 15	E0, F0, 15
Stop	E0, 3B	E0, F0, 3B
Play/Pause	E0, 34	E0, F0, 34
Mute	E0, 23	E0, F0, 23
Volume Up	E0, 32	E0, F0, 32
Volume Down	E0, 21	E0, F0, 21
Media Select	E0, 50	E0, F0, 50
E-Mail	E0, 48	E0, F0, 48
Calculator	E0, 2B	E0, F0, 2B
My Computer	E0, 40	E0, F0, 40
WWW Search	E0, 10	E0, F0, 10
WWW Home	E0, 3A	E0, F0, 3A
WWW Back	E0, 38	E0, F0, 38
WWW Forward	E0, 30	E0, F0, 30
WWW Stop	E0, 28	E0, F0, 28
WWW Refresh	E0, 20	E0, F0, 20

ПРИЛОЖЕНИЕ 2. ЗАДАНИЯ НА ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа № 1. «Организация современных персональных компьютеров»

Написать реферат на одну из приведённых ниже тем. Объем реферата должен быть таким, чтобы максимально полно осветить тему. Реферат должен содержать следующие разделы:

- титульный лист;
- содержание;
- введение;
- основной текст;
- заключение;
- список использованной литературы, включая электронные источники.

Темы рефератов

1. Видеоадаптеры (видеокарты)
2. Сетевые карты
3. Манипуляторы типа «мышь»
4. Мониторы
5. Звуковые карты
6. Модемы
7. Последовательный интерфейс
8. Параллельный интерфейс
9. Память DIMM
10. Память RIMM
11. Интерфейс CNR
12. Интерфейс AMR
13. Интерфейс IEEE1394
14. Стандарт Wi-Fi
15. Стандарт Bluetooth
16. Приводы для чтения и записи компакт дисков и DVD.

Контрольные вопросы

1. Что такое ЭВМ? Персональный компьютер?
2. Зачем нужна материнская плата?
3. Зачем используется блок питания? Корпус?
4. Что такое набор микросхем системной логики?
5. Что такое форм-фактор?
6. Сколько шин в персональном компьютере? Зачем они нужны? Как определить пропускную способность шины?
7. Виды памяти? Статическая и динамическая память?
8. Что такое интерфейс? Какие интерфейсы используются в ПК?

Лабораторная работа № 2. «Элементы цифровой логики и булевой алгебры. Регистры флагов. Системы счисления»

1. Согласно варианту задания построить таблицы истинности, получить алгебраические формулы булевых функций и синтезировать логические схемы двух комбинационных устройств. Схемы устройств синтезируются по раздельности.
2. Разработать программу на языке Си, реализующую модель АЛУ, вычисляющую значения логических функций для указанных устройств. Значение входов АЛУ (два операнда и код операции) вводятся пользователем с клавиатуры в указанной системе счисления. Далее эти значения записываются в одну целую переменную, в которой 4 младших разряда (0-3) используется для записи первого операнда, следующие 4 (4-7) – для значения второго операнда, следующий бит (8)– для значения кода операции. Полученная переменная передаётся (по указателю) в функцию ALU, которая выполняет указанную операцию над операндами и записывает результат в разряды с номерами (9-13). Другими словами функция ALU рассчитывает значения разрядов (9-13) согласно полученным в п.1 функциям. При этом разряды 9-12 используются для значения результата, а 13-й разряд - для сигнала о переполнении или произошедшей ошибке в работе АЛУ. После выполнения функции ALU результат (т.е. вся полученная переменная) выводится в двоичной системе счисления на экран. Для связи устройств внутри ALU должны быть приняты следующие правила:
    - мультиплексор для входов D использует значения первого операнда, для входов A – второго, значение выхода Y записывается в младший разряд на выходе;
    - демультиплексор – для входа Y использует значение младшего бита первого операнда, для входов A – два младших разряда второго операнда;
    - шифратор для входов D использует первый операнд, значение второго операнда игнорируется, результат записывается на выходе в три младших разряда;
    - дешифратор – для входов A использует три младших разряда первого операнда, значение второго операнда игнорируется;
    - сумматор – для входов X использует значения первого операнда, для входов Y – второго;
    - компаратор – для входов X использует значения первого операнда, для входов Y – второго, на выходе результат записывается в младший разряд;
    - сдвиговый регистр – для входов X использует первый операнд, для входа C – младший разряд второго операнда.

Варианты задания

№	Схемы	Система счисления
1	Мультиплексор, компаратор	Десятичная
2	Демультиплексор, компаратор	Десятичная
3	Шифратор, компаратор	Десятичная
4	Дешифратор, компаратор	Десятичная
5	Мультиплексор, сумматор	Восьмеричная
6	Демультиплексор, сумматор	Восьмеричная
7	Шифратор, сумматор	Восьмеричная
8	Дешифратор, сумматор	Восьмеричная
9	Мультиплексор, сдвиговый регистр	Шестнадцатеричная
10	Демультиплексор, сдвиговый регистр	Шестнадцатеричная
11	Шифратор, сдвиговый регистр	Шестнадцатеричная
12	Дешифратор, сдвиговый регистр	Шестнадцатеричная

Контрольные вопросы

1. Что такое вентиль? Какие значения он может принимать?
2. Сколько вентилей необходимо, чтобы получить логические функции НЕ, ИЛИ-НЕ, И-НЕ, И, ИЛИ?
3. Что такое таблица истинности? Булева функция? Как они связаны между собой?
4. Как получить алгебраическую булеву функцию из таблицы истинности? И наоборот?
5. Каким образом можно синтезировать логическую схему по таблице истинности? По алгебраической формуле?
6. Что такое система счисления? Чем отличается позиционная система счисления от непозиционной?
7. Как получить качественный эквивалент числа в непозиционной системе счисления? В позиционной?
8. Как перевести числа из двоичной системы счисления в десятичную? Восьмеричную? Шестнадцатеричную? И наоборот?
9. Что такое двоично-десятичное число?
10. Как в ЭВМ представляются отрицательные числа и числа с плавающей запятой?
11. Что такое дополнительный код? Зачем он используется?
12. Как перевести десятичное число с плавающей запятой в двоичное?
13. Какие базовые типы данных используются для хранения переменных в языке СИ?
14. Что такое флаг? Зачем он используется? Каким образом можно манипулировать флагами? Что такое маска?

Лабораторная работа № 3. «Представление текстовой информации в ЭВМ. Терминалы».

1. Используя команду infocmp получите управляющие последовательности, доступные Вашему терминалу.
2. Получив список команд, разработать функции[1] для:
   a) очистки экрана
   b) перемещение курсора в указанную позицию экрана
   c) изменения цвета фона и текста
   d) вывода символа псевдографики
   e) вывода псевдографической рамки указанных размеров
3. Используя команду read, получите коды (или управляющие последовательности) для клавиш управления курсором, F1, F12.
4. Разработайте функцию, настраивающую терминал таким образом, чтобы он стал работать в неканоническом режиме без отображения вводимых символов на экран, и ожидал получения с терминала указанного количества символов в течение указанного времени.
5. Используя разработанные функции, напишите программу, реализующую алгоритм согласно варианту задания.

Вариант	Задание
1	Написать программу вывода на экран бегущей строки. Изначально текст бежит по средней строке (число строк / 2) справа налево. Реализовать возможность управления движением текста так, чтобы можно было изменять строку, по которой движется текст (двигать текст вверх и вниз) путем нажатия клавиш управления курсора. Выход осуществляется по нажатию клавиши ESC
2	Написать программу вывода на экран увеличенных символов, введённых с клавиатуры. В качестве допустимых символов взять следующие: T, Y, I, L, F, Z. Каждый символ кодируется растром. В случае нажатия недопустимой клавиши экран очищается и в верхнем левом углу экрана выводится ФИО и группа автора программы. Выход осуществляется по нажатию клавиши ESC. Если пользователь не нажал на клавишу больше, чем 3 секунды, то экран очищается и посередине выводится информация об авторе программы.
3	Написать программу движения квадрата (3 на 3) по экрану. При нажатии клавиши w размер по горизонтали увеличивается, при нажатии на q размер по горизонтали уменьшается, при нажатии на e размер по вертикали увеличивается, при нажатии на d размер по вертикали уменьшается. Границы экрана должны быть обозначены псевдографической рамкой
4	Написать программу, реализующую простейшую игру по заполнению. «стакана» символами. Алгоритм программы следующий: на экране

[1] Для тех студентов, кто претендует на «автомат» на экзамене, эти функции необходимо оформить в виде статически подключаемой библиотеки и заголовочного файла. Окончательная программа должна использовать заголовочный файл для компиляции и саму библиотеку, подключая её в момент линковки.

	суется прямоугольник («стакан»). В середине верхней строки жается символ «а». Если пользователь не нажал клавишу в течении ной секунды, то символ опускается на одну строку вниз и так до тех пор, пока не будет достигнут конец «стакана» или строка, уже ненная символом. Если пользователь в процессе «падения» символа нажмет клавиши управления курсором «стрелка вверх» и «стрелка вниз», то символ изменится на следующий из множества: «а», «б», «в», «м» и «л». Если пользователь в процессе «падения» символа нажмет клавиши управления курсором «стрелка вправо» и «стрелка влево», то позиция символа изменяется соответственно на один столбец влево и вправо. При достижении символом дна «стакана» проверяется дение 5 символов по диагонали, вертикали и горизонтали и, в случае совпадения, символы удаляются со смещением находящихся выше символов вниз. Игра заканчивается после того, как пользователь нажмет клавишу ESC или стакан переполнится хотя бы по одному из столбцов.
5	Написать простейшую игру «Арканоид». По экрану движется символ W. Дойдя до верхней, либо правой, либо левой, либо нижней границы символ «отталкивается» от них. От нижней границы символ отталкивается только в том случае, если в этом месте находится горизонтальная площадка, управляемая пользователем. Если пользователь не успел подвести площадку под символ, игра

Контрольные вопросы

1. Схема и подключение клавиатуры к компьютеру.
2. Типы клавиш.
3. Взаимодействие с устройствами в Linux. Специальные файлы устройств.
4. Функции open, close, read, write.
5. Терминалы. Типы терминалов. Эмуляция терминала. Режимы работы.
6. Управление терминалом. Команды. Низкоуровневое управление.
7. Структура termios. Состав и назначение.
8. Состав и назначение базы terminfo.

Лабораторная работа № 4. «Подсистема прерываний ЭВМ. Сигналы и их обработка»

Доработать программу из третьей лабораторной работы согласно варианту задания.

Вариант	Задание
1	Изменить программу таким образом, чтобы позиция строки по горизонтали (по столбцам) изменялась по сигналу от таймера. При нажатии на клавишу ‘g’ скорость движения строки (частота срабатывания таймера) должна увеличиваться, а при ‘h’
2	Реализовать обработчик сигнала от драйвера терминала об нии размеров окна. Символы должны выводиться всегда в середине экрана. Если размер экрана меньше, чем размер символов, то

	жен сгенерироваться сигнал SIGUSR1 по которому программа должна очистить экран, вывести информацию об авторе программы (ФИО и группу) и завершиться.
3	Организовать движение квадрата по сигналу от таймера. Скорость движения квадрата изменяется нажатием клавиш ‘g’ (ускоряется) и ‘h’ (замедляется). Клавиши управления курсором должны изменять направление движения квадрата.
4	Изменить программу таким образом, чтобы вертикальная позиция символа изменялась по сигналу от таймера.
5	Изменить программу таким образом, чтобы позиция символа изменялась по сигналу от таймера.

Контрольные вопросы

1. Что такое прерывание? Что такое сигнал? Чем они отличаются друг от друга? Какую информацию несут в себе прерывание и сигнал?
2. Как происходит обработка сигнала в программах, работающих под управлением ОС Linux?
3. Каким образом настраивается таймер? Как программа «узнаёт» о срабатывании таймера?
4. Каким образом пользовательская программа может узнать об изменении размера окна виртуального терминала?

Лабораторная работа № 5. «Устройство хранения данных на жестких магнитных дисках»

Написать программу, позволяющую сформировать таблицы разделов на жестком диске.

Программа должна выполнять следующие действия:

1. При запуске (модель загрузки ПК) проверяется наличие в текущем каталоге файла с именем DISK.txt, который содержит информацию о геометрии жесткого диска. Формат файла следующий: каждая строка содержит запись вида «Параметр<пробел>=<пробел>значение». Например, «C = 7865». Если такого файла нет, или его структура не соответствует указанной, или в файле представлены не все параметры, то на экран выдаётся сообщение об ошибке, информация об авторах программы, и работа программы завершается. При успешном чтении файла, на экран выводится информация о диске (геометрия, размер в Гбайт).
2. При загрузке программы считается, что диск пустой (т.е. на нем нет никакой информации). Программа спрашивает пользователя, будет ли он добавлять раздел? Если пользователь намеревается добавить раздел, то ему предлагается ввести информацию о требуемом размере и типе раздела. После каждого добавления выводится на экран таблица уже введённых значений. Изначально эта таблица пустая. Формат таблицы: номер, тип раздела, размер.
3. После того, как пользователь ввел информацию обо всех требуемых ему разделах, программа формирует необходимые таблицы разделов, включая расширенные, и выводит их в файл с именем TABLES.txt. При выводе расширенных таблиц указывается номер сектора, в котором они должны располагаться. Счита-

ется, что разделы начинаются с 1 сектора, первые три раздела будут основными, остальные – логическими и будут располагаться в расширенном разделе.

Контрольные вопросы

1. Основные этапы загрузки ПК на базе процессоров семейства Intel.
2. Зачем используется сигнал “RESET”?
3. Магнитные диски. Зачем используются. Устройство.
4. Магнитные головки чтения/записи. Типы. Зачем используются. Принцип работы.
5. Привод магнитных головок. Типы приводов. Зачем используются.
6. Контроллер управления. Зачем используется.
7. Геометрия. Что это такое? Трансляция геометрии. Типы трансляции.
8. LBA адресация. Зачем используется. Перевод из LBA в CHSлог и наоборот.
9. Барьеры размеров дисков. Почему возникли? Какие присутствуют?
10. Этапы загрузки ПК.
11. Логическая организация винчестера. Разделы диска. Таблица разделов. Зачем используется. Структура.

Сергей Николаевич Мамойленко
Организация ЭВМ и систем
Практикум

Редактор: Ю.С. Майданов.
Корректор: Д.С. Шкитина.

Подписано в печать _____ Формат бумаги 60 х 84/16, отпечатано на ризографе, шрифт № 10, изд. л. 6,3 заказ №_____, тираж – 100 экз., СибГУТИ.
630102, г. Новосибирск, ул. Кирова, д. 86