ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «СИБИРСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

**С.Н. Мамойленко**
**О.В. Молдованова**

# ЭВМ и ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА

Учебное пособие

Новосибирск
2012

УДК 681.3.068(075)

Мамойленко С.Н., Молдованова О.В. ЭВМ и периферийные устройства: Учебное пособие. – Новосибирск: СибГУТИ, 2012. – 106 с.

Учебное пособие предназначено для студентов, обучающихся по направлению подготовки 230100 «Информатика и вычислительная техника» и изучающих дисциплину «ЭВМ и периферийные устройства». 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. – 46, табл. – 5, список лит. – 7 наим.

Рецензент: доцент кафедры прикладной математики и кибернетики ФГОБУ ВПО «СибГУТИ» Перцев И.В.

Для студентов, обучающихся по направлению 230100 «Информатика и вычислительная техника».

Утверждено редакционно-издательским советом ФГОБУ ВПО «СибГУТИ» в качестве учебного пособия.

© Мамойленко С.Н., Молдованова О.В., 2012

© ФГОБУ ВПО «СибГУТИ», 2012

ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

**Электронная 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» особое внимание уделяется изучению принципов построения вычислительных машин и систем. Согласно государственному образовательному стандарту первое знакомство с этими принципами студенты получают в рамках курса «ЭВМ и периферийные устройства», отнесённого к базовой части (Б.3) профессионального блока дисциплин. Очевидно, что ограничиваться только получением теоретических навыков для студентов нецелесообразно. Важным является организация практических занятий, помогающих студентам закрепить полученные теоретические знания.

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

В учебном пособии представлен материал 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 для лабораторных работ, выполняемых в рамках курса «ЭВМ и периферийные устройства» студентами, обучающимися на Кафедре вычислительных систем ФГОБУ ВПО «Сибирский государственный университет телекоммуникаций и информатики».

# ГЛАВА 1. КРАТКАЯ ИСТОРИЯ ВОЗНИКНОВЕНИЯ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

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


Рисунок 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 ёмкостью 256 байтов (!). 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 использованием 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 Computers выпустила свою модель персонального компьютера – Apple I (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 (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 ещё больше увеличил популярность этого ПК.

Рисунок 2. Компьютер Apple I

Рисунок 3. Компьютер Apple II

Рисунок 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 Льюис 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 (рис. 5) появился 12 августа 1981 года и стал родоначальником нового стандарта построения ЭВМ.

Рисунок 5. Первый IBM PC

Рисунок 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.

## Контрольные вопросы

1. Расскажите об истории возникновения персонального компьютера.
2. Что представляла собой ЭВМ Altair-8800?
3. Когда была выпущена модель персонального компьютера Apple I?
4. В чём главные отличия Apple II от Apple I?
5. Какую цель ставили перед собой создатели лаборатории Entry System Division?
6. Когда появился первый IBM PC?
7. Какие изменения претерпели ПК за свою двадцатилетнюю историю?

# 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 (жёсткий 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** (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, объединяющие 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-фактором 5,25 дюйма[1] (дисководы магнитных дисков, CD-ROM и т.д.). Кроме этого, существуют узкие (англ. slim) корпусы башенного типа. В таких корпусах отсеки размером 5,25" располагаются вертикально, а ширина корпуса определяется устройствами с форм-фактором 3,5" (дисководы, ZIP-диски и т.п.).

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

[1] Дюйм – единица измерения, равная 2,54 см. Для обозначения дюйма используют двойной штрих (").

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

Рисунок 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 (рис. 9) представляет собой многослойную плату, объединяющую все электрические схемы, которые, в совокупности, и образуют вычислительную машину. Кроме электрических схем на ней располагаются разъёмы для подключения процессора, памяти и периферийных устройств, а также некоторые переключатели. Структуру системной платы определяют состав смонтированных на ней 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, это далеко не полный их перечень.

Рисунок 8. Разъёмы 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 разъём клавиатуры типа DIN

(круглый 5-штырьковый). Для подключения блока питания используется 12-штырьковый разъем (рис. 8), состоящий из двух частей (по 6 линий каждый). Устанавливать его надо так, чтобы все чёрные 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 разъёмы типа PS/2 (круглый 6-штырьковый) для клавиатуры и «мышки» и USB для подключения внешних устройств (рис. 9). Введены дополнительные каналы для взаимодействия с блоком питания. Для подключения последнего используется один 20-контактный разъём. 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 (англ. Extended ATX), но она не получила широкого распространения и была вытеснена форм-фактором WTX (англ. Workstation Technology eXtended).

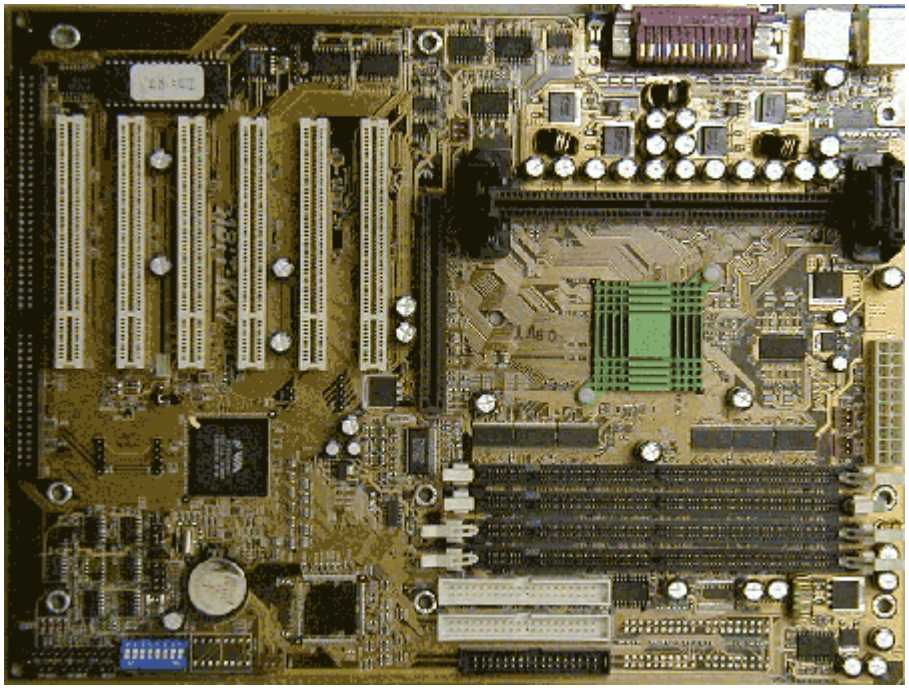


Рисунок 9. Пример системной платы (форм фактор FullATX)

В узких (англ. slim) корпусах стал применяться **форм-фактор LPX**. Сокращение размеров было реализовано путём 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 персонального 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, разделённая на две части: часть, где находятся собственно системные компоненты, и подсоединённая к ней через 340-контактный разъём 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,** ориентированный 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 подключёнными к имеющимся на ней разъёмам внешними устройствами, ещё не является вычислительной машиной. Для организации взаимодействия между всеми этими узлами используется набор микросхем системной логики, называемый также **чипсетом** (англ. chipset). 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 (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 процессор, оперативная память, ускоренный графический порт AGP (англ. Accelerated Graphics Port), интерфейс с внешними устройствами PCI (англ. Peripheral

Component Interconnect, 33 МГц). Чаще всего он работает на полной тактовой частоте системной платы (на частоте шины процессора). В функции северного моста входит управление потоками данных из 4-х шин: шины памяти, AGP, системной шины процессора и шины связи с южным мостом. В современных наборах микросхем северный мост реализован в виде одной микросхемы (на одном кристалле), раньше для его реализации требовалось до трёх микросхем.

Южный мост (англ. south bridge) обслуживает медленные устройства, подключаемые через шину ISA (8 МГц), контроллер жёсткого диска IDE и USB (Universal Serial Bus). Также он содержит в себе схемы, реализующие функции энергонезависимой памяти системы ввода-вывода BIOS (англ. Complementary Metal–Oxide–Semiconductor, CMOS) и часов реального времени, контроллер прямого доступа к памяти 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.).

Базовая система ввода-вывода содержит программу начального тестирования POST (Power On Self-Test), программу начальной загрузки, драйверы для интегрированных устройств.

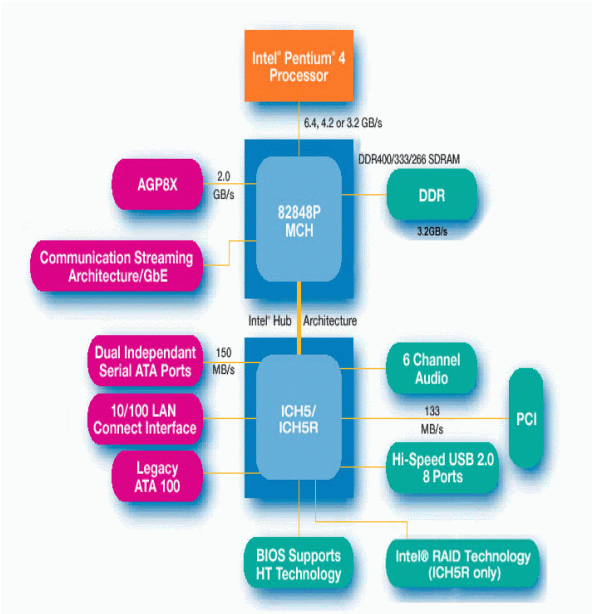



Рисунок 10. Набор микросхем системной логики (Intel 845)

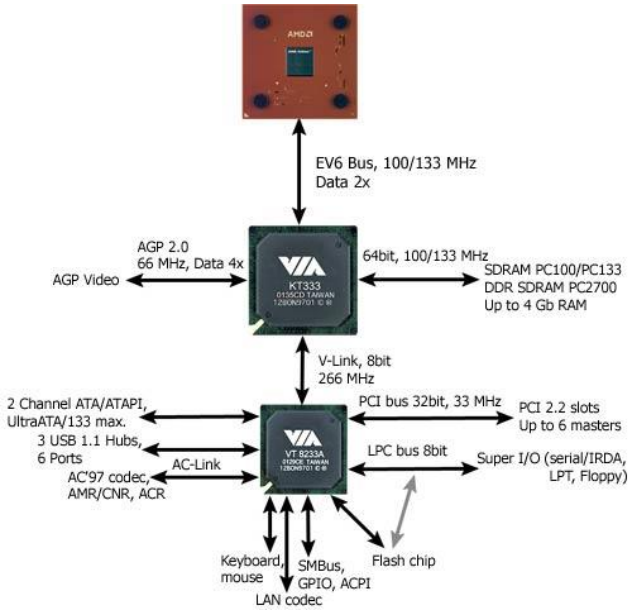



Рисунок 11. Набор микросхем системной логики (VIA Apollo Pro)

С недавнего времени (с выпуском чипсета i815) корпорация Intel отказалась от использования архитектуры мостов (рис. 12) и перешла к похожей архитектуре, в которой используются концентраторы – хабы (англ. hub). На первый взгляд всё то же самое: два концентратора, один был раньше северным мостом, а теперь называется «хаб контроля памяти» (англ. Memory Controller Hub), другой был южным мостом и называется «хаб контроля ввода-вывода» (англ. I/O Controller Hub). Функции концентраторов не поменялись, однако они стали более независимы, а интерфейс их связи друг с другом представляет собой связь «один-к-одному» (англ. point-to-point) через специальный концентра-

тор (раньше мосты соединялись через шину PCI, имеющую вдвое меньшую скорость, чем концентратор).

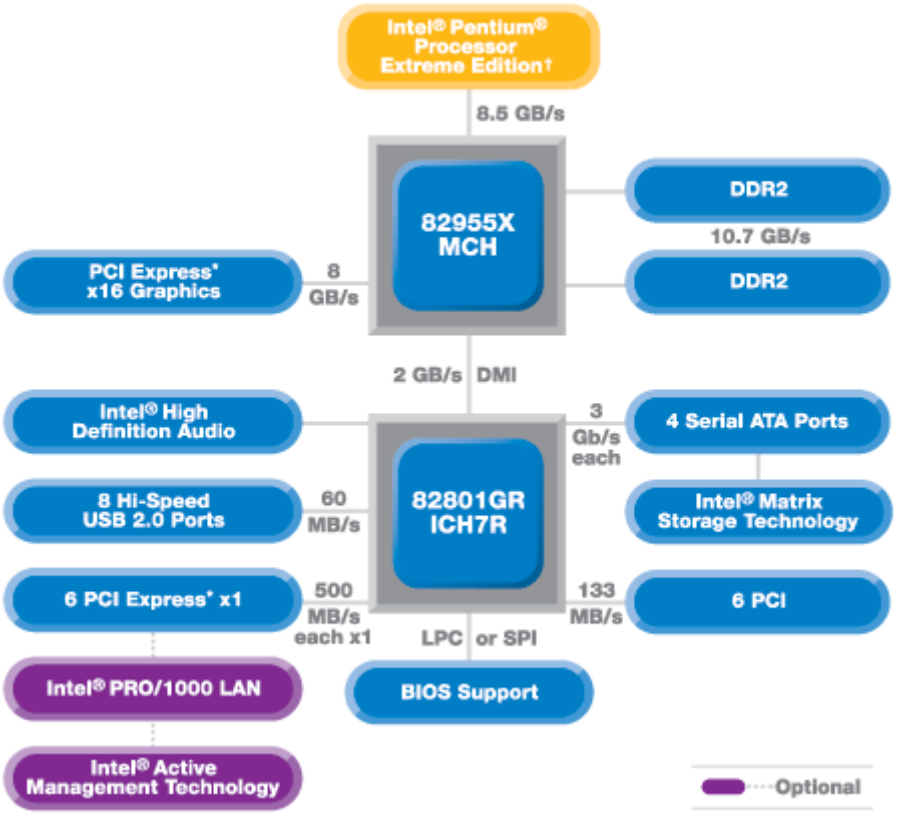



Рисунок 12. Набор микросхем системной логики (Intel 945)

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 данных, шина адреса и шина управления). При этом в один момент времени передавать информацию по шине может только одно устройство, а все остальные подключённые 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 определённый интервал времени посылается сигнал напряжения высокого уровня (обычно это +5 В), а для передачи нулевого бита данных – сигнал напряжения низкого уровня (обычно это 0 В).

Чем больше линий в шине (т.е. чем больше её разрядность), тем больше битов можно передать за одно и то же время. Таким образом, *скорость передачи данных по шине* (или её пропускная способность) определяется как произведение её тактовой частоты на разрядность.

Аналогом компьютерной шины можно назвать обыкновенную электрическую сеть, в которой передающим устройством является электростанция, а принимающими устройствами являются все подключённые 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 путём 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 гнёзда для подключения внешних устройств

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

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

EISA (англ. Enhanced ISA – расширенная шина ISA) – функциональное и конструктивное расширение ISA. Внешне разъёмы имеют такой же вид, как и у ISA, и в них могут вставляться платы ISA. Но в глубине разъёма находятся дополнительные ряды контактов EISA, а платы EISA имеют более высокую ножевую часть разъёма 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 МБ/с.

VLB (англ. VESA Local Bus – локальная шина стандарта VESA) – 32-разрядное дополнение к шине ISA. Конструктивно представляет собой дополнительный разъём (116-контактный) при разъёме ISA. Разрядность – 32 бита,

тактовая частота – в диапазоне от 25 до 50 МГц. PCI (англ. 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 МБ/с (для 64-разрядной шины на 66 МГц).

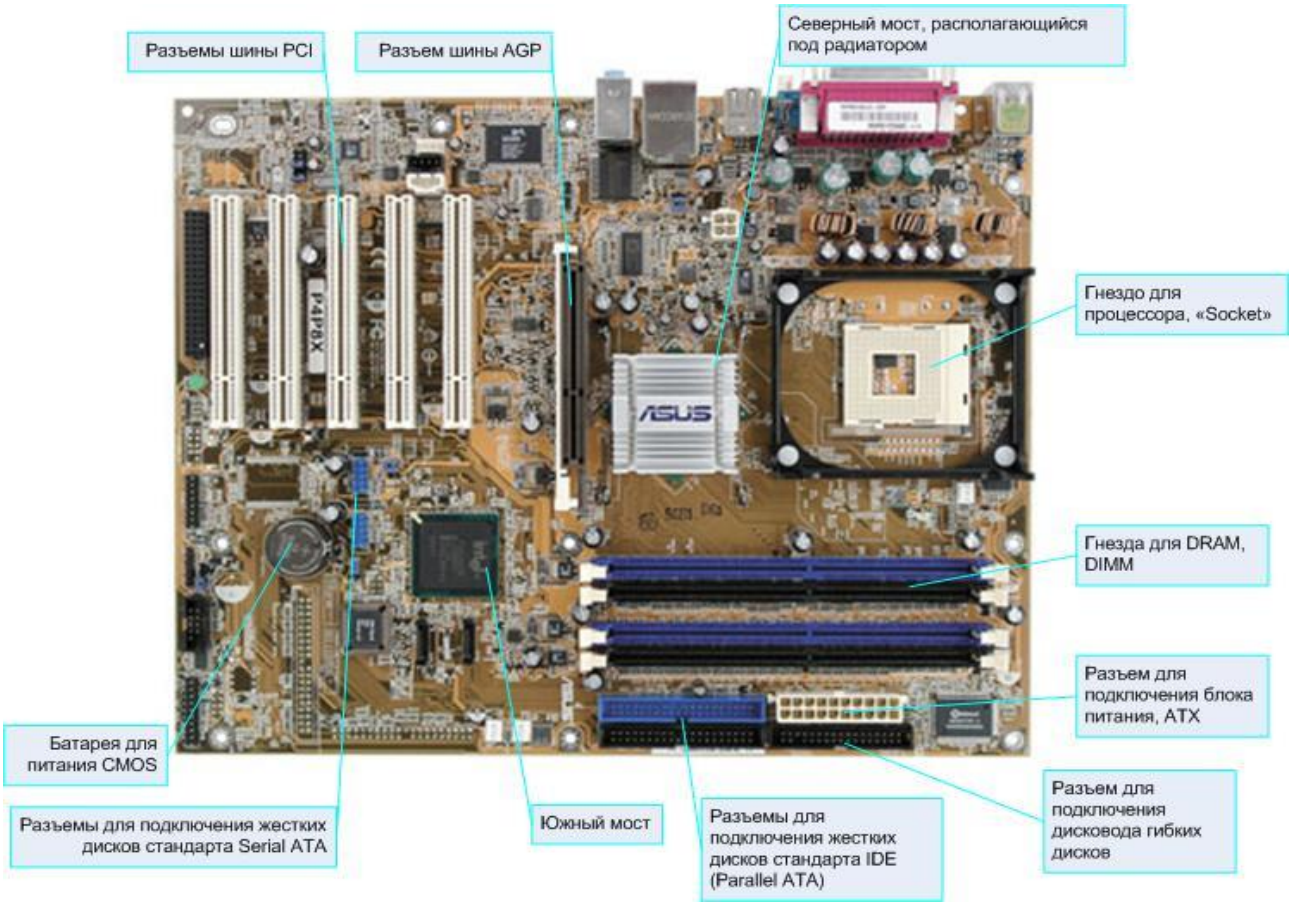



Рисунок 13. Элементы материнской платы на примере ASUS P4P8X


Рисунок 14. Внешние разъемы на примере материнской платы ASUS P4P8X

AGP (англ. Accelerated Graphics Port – ускоренный графический порт) является дальнейшим развитием PCI, нацеленным на ускоренный обмен графическими адаптерами с оперативной памятью. Пропускная способность увеличена за счёт 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. определённая последовательность нулей и единиц). Для подключения устройств используются 9- или 25-штырьковые разъёмы. 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-штырькового разъёма. Существуют одно- и двунаправленные параллельные интерфейсы.

PS/2 порты. Практически полный аналог COM-порта. Служат для подключения клавиатуры или манипулятора «мышь».

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

FDD (англ. Floppy Disk Driver – накопитель на гибких магнитных дисках). Конструктивно представляет собой 12x2-контактный игольчатый разъём с возможностью подключения двух дисководов через соединительный кабель – шлейф. В один момент времени информацию может передавать только один дисковод.

IDE (англ. Integrated Drive Electronics) или ATA (англ. AT Attachment). Используется для подключения устройств хранения информации (жёсткого диска, CD-ROM и т.п.). Конструктивно представляет собой 20x2-контактный игольчатый разъём, 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 – 33 МБ/с и UDMA/66 – 66 МБ/с. Протокол UDMA/66 обладает вдвое большей скоростью передачи данных за счёт того, что данные передаются по обоим фронтам тактирующего сигнала в отличие от UDMA/33, вследствие чего необходим шлейф, в котором бы отсутствовали помехи от 2-х параллельно идущих проводников. Для решения этой проблемы применяется 80-жильный шлейф, каждый второй проводник которого соединён с общим проводом для уменьшения помех.

## 2.5. Гнёзда для подключения процессоров

Для установки процессоров на системную плату используются гнёзда, 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.

Гнёзда горизонтального типа представляют собой прямоугольный пластмассовый планшет с отверстиями, в котором располагаются металлические разъёмы. Число отверстий зависит от типа сокета. Первые версии таких гнёзд 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 гнёзд 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 разъёмах 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 собой пластиковый разъём 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, жёсткие 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 четвёртого 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, который 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 своё содержимое бесконечно долго, пока не будет выключено электропитание. За счёт отсутствия циклов регенерации и высокой скорости доступа к ячейкам статическая память значительно быстрее, чем динамическая. Однако триггеры намного 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 разъёмы 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 разъёмов для динамической памяти: SIMM (англ. Single Inline Memory Module – одинарный модуль памяти), DIMM (англ. Dual Inline Memory Module – двойной модуль памяти), RIMM (англ. 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. Он вставляется в разъём под углом и поворачивается до вертикального положения, зажимаясь при этом держателями, расположенными по краям разъёма. 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 объём 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 разъём, 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 220 В / 50 Гц или 120 В / 60 Гц. Электронными схемами используется постоянное напряжение в + 3,3 В, ± 5 В и ± 12 В.

С внешней стороны (рис. 15) блок питания имеет разъёмы для кабеля, подключаемого к электрической сети, и для кабеля, подключаемого к другим внешним (по отношению к системному блоку) устройствам, например, к монитору, звуковым динамикам и т.п. Второй разъём может отсутствовать.

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

Рисунок 15. Блок питания ПК

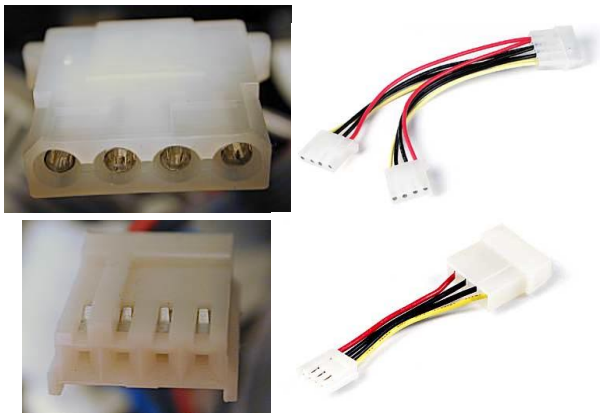


Рисунок 16. Разъёмы для подключения внутренних устройств

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, подключаемый к специальному разъёму на материнской плате и также располагающийся на лицевой стороне корпуса.

В первом случае для подключения к материнской плате используется 12-ти штырьковый разъём, во втором случае – 20-ти 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).

## Контрольные вопросы

1. Назовите основные составные части современного ПК.
2. Что такое форм-фактор? Назовите форм-факторы для корпусов ПК.
3. Для чего нужна системная (материнская) плата? Назовите форм-факторы материнских плат.
4. Назовите отличия форм-фактора ATX материнской платы от форм-фактора AT.
5.Что такое чипсет?
6. Объясните назначение северного и южного мостов.
7. Что такое шина? Сколько шин в ПК?
8. Как происходит передача данных по шине? Что такое разрядность шины?
9. Как определить пропускную способность шины?
10. Что такое системная тактовая частота?
11. Что такое интерфейс? Какие интерфейсы используются в ПК?
12. Какие устройства подключаются через последовательный порт, а какие – через параллельный?
13. В чем отличие сокета от слота?
14. Какие виды запоминающих устройств используются в современных ПК? Какие из них являются энергозависимыми, а какие – нет?
15. Для чего используются статическая и динамическая память? Расскажите о типах разъёмов для динамической памяти.
16. Объясните назначение блока питания в ПК.

# ГЛАВА 3. ПРОЦЕДУРА ЗАГРУЗКИ ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА

Первым действием, которое выполняет компьютер при включении питания, является процедура **загрузки**, т.е. последовательность действий аппаратурной части ПК по проверке состава (наличия или отсутствия) 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 регистров арифметико-логического устройства устанавливаются в нулевое значение.

С момента снятия сигнала *«RESET»* микропроцессор начинает работу в реальном режиме, и, в течение примерно 7-ми циклов синхронизации, приступает к выполнению инструкции, считываемой из ROM BIOS и располагающейся 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 (англ. Power On Self-Test). На данном этапе тестируются процессор, память и системные средства ввода-вывода, а также производится конфигурирование программно управляемых аппаратурных средств материнской платы. Обнаружив ошибку, система подаёт определённый 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 на экран выводится 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. Этот драйвер пытается стандартным образом инициализировать видеоадаптер и вывести на экран курсор. Если и это не срабатывает, то видеоадаптер считается неисправным, и подаётся соответствующий звуковой сигнал.

Далее сканируется память по адресам с C800:0000 по DF80:0000 с шагом 2 КБ в поисках встроенных драйверов любых других подключённых 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 подключённые устройства. Часть конфигурирования выполняется однозначно, а другая часть может определяться положением перемычек (переключателей) системной платы или содержимым энергонезависимой памяти CMOS. Для изменения CMOS используется встроенная в BIOS утилита, называемая «Setup».

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 (англ. boot sequence) внешних устройств, чтобы загрузить операционную систему (ОС, 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 эта информация программой дальнейшей загрузки ОС. Если значения первых байтов считанного блока данных не корректны, 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.

В случае корректности считанного блока данных начинается загрузка операционной системы, процедура которой зависит от типа ОС.

### Контрольные вопросы

1. В чём состоит процедура загрузки ПК?

2. В какие состояния устанавливаются сегментные регистры и регистр-указатель команд процессора при загрузке ПК?
3. Что такое POST?
4. Какое сообщение выдаётся при отсутствии встроенного драйвера видеоадаптера?
5. Какое значение хранится по адресу 0000:0472?
6. Что такое «горячая» перезагрузка? Что такое «холодная» перезагрузка?
7. Как определяется порядок поиска устройства для загрузки операционной системы?

# 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 электронные 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 путём арифметических операций над эквивалентами цифр.

В зависимости от того, меняет ли свое количественное значение цифра при разном положении в числе, системы счисления можно классифицировать как *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 озна-

чает единицу, V – пять, X – десять, L – пятьдесят, C – сто, D – пятьсот, M – тысячу. Для получения числа требуется просто просуммировать количественные эквиваленты входящих в него цифр, с учётом 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 неестественно. 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 (англ. Binary Coded Decimal, BCD).

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 ($1 \cdot 10^1 + 0 \cdot 10^0$). Эта система используется нами в повседневной жизни. Такое распространение она получила из-за того, что первоначально счёт 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 ($7 \cdot 8^1 + 7 \cdot 8^0$), а $10_{(8)}$ равно восьми ($1 \cdot 8^1 + 0 \cdot 8^0$).

В *шестнадцатеричной системе счисления* набор цифр включает арабские цифры от 0 до 9 и 6 букв латинского алфавита – A (десять), B (одиннадцать), C (двенадцать), D (тринадцать), E (четырнадцать), F (пятнадцать). Например, число $FF_{(16)}$ имеет количественный эквивалент двести пятьдесят пять

$(15 \cdot 16^1 + 15 \cdot 16^0)$, а $100_{(16)}$ равно двумстам пятидесяти шести $(1 \cdot 16^2 + 0 \cdot 16^1 + 0 \cdot 16^0)$.

*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 как упакованное – $0011\ 0100\ 0101\ 0110_{(2)}$.

### *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 путём деления исходного числа на основание требуемой системы и записи остатков от деления в обратном порядке. Например, смотрите рисунок 17.

Перевод числа из шестнадцатеричной системы в двоичную систему может быть осуществлён путём 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 путём 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 $347_{(8)}$.

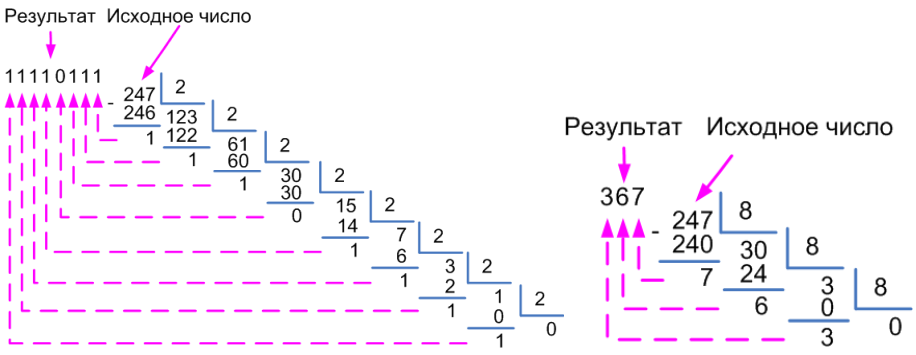



Рисунок 17. Перевод числа 247 из десятичной системы счисления в двоичную (слева) и восьмеричную (справа) системы

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 $377_{(8)}$.

### *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 чисел от положительных.

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. Причём инвертируется вся ячейка памяти, отведённая под число (будем считать, что мы работаем с 16-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 старший 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 модуль в $q$-ичном коде. Здесь $q$ – это *основание системы* или *база*. В ЭВМ чаще всего используется двоичная система, однако существуют ещё 8- и 16-ричные 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. 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 (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.

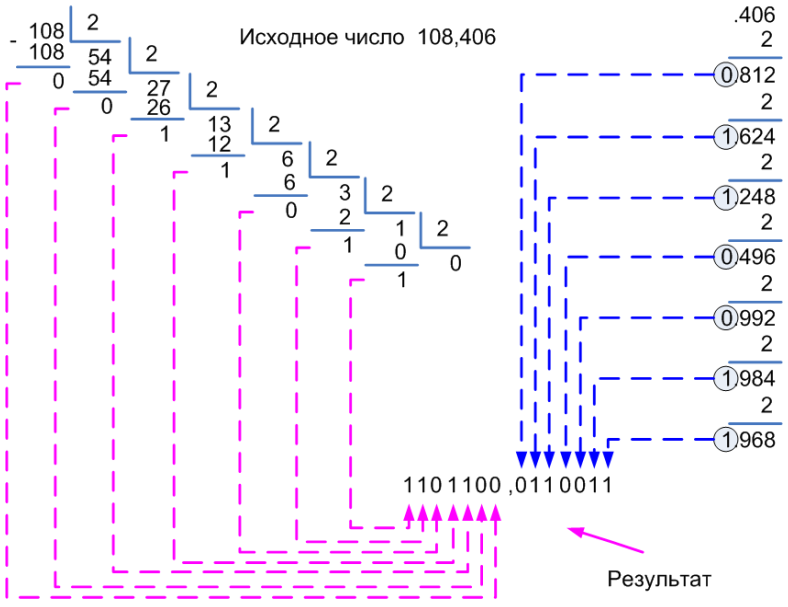



Рисунок 18. Перевод числа 108,406 из десятичной системы в двоичную систему счисления

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-ю разрядами под порядок. Первый тип называется одинарным или простым, второй – двойным или 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 к порядку расположения специальных символов (символов национальных языков, знаков препинания, математических символов и т.п.) обычно не предъявляется.

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

Таблица ASCII делится на две части: базовую и расширенную. 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 чётности (т.е. принимал единичное значение, если в 7-битной части чётное число единиц). Основная таблица описывает 128 символов, из которых (рис. 19):

- первые 32 кода отданы производителям аппаратурных 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.

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-битных кодировок, где младшая половина каждой кодировки (символы с кодами 0–127) соответствует ASCII, а старшая половина определяет символы для различных язы-

ков. В зависимости от использования кодов 128–255 различают следующие вариации стандарта ISO 8859 (табл. 1).

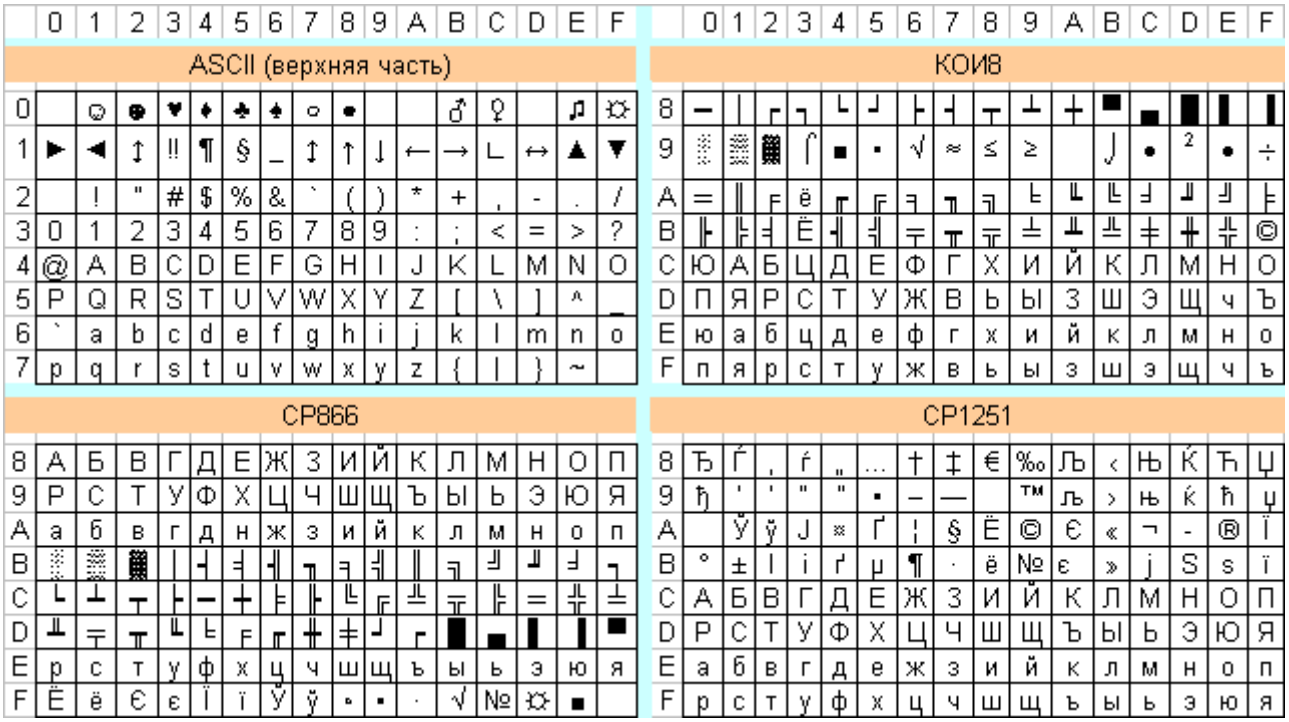



Рисунок 19. Таблицы символов

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 идиш |
| ISO 8859-9 | Турецкий язык |
| 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 KOI8, 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 KOI8, убрать восьмой бит каждого символа, то текст останется читаемым, хотя и выглядеть будет забавно. Например, слово «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) (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, KOI8 и OEM хотя и предназначены для одного и того же (кодирования символов национальных языков), но не согласованы между собой. Поэтому при передаче информации необходимо чётко 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 Computers, 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 он даёт возможность одновременно представлять символы всех «живых» и «мёртвых» 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 определяется вид соответствующего символа.

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

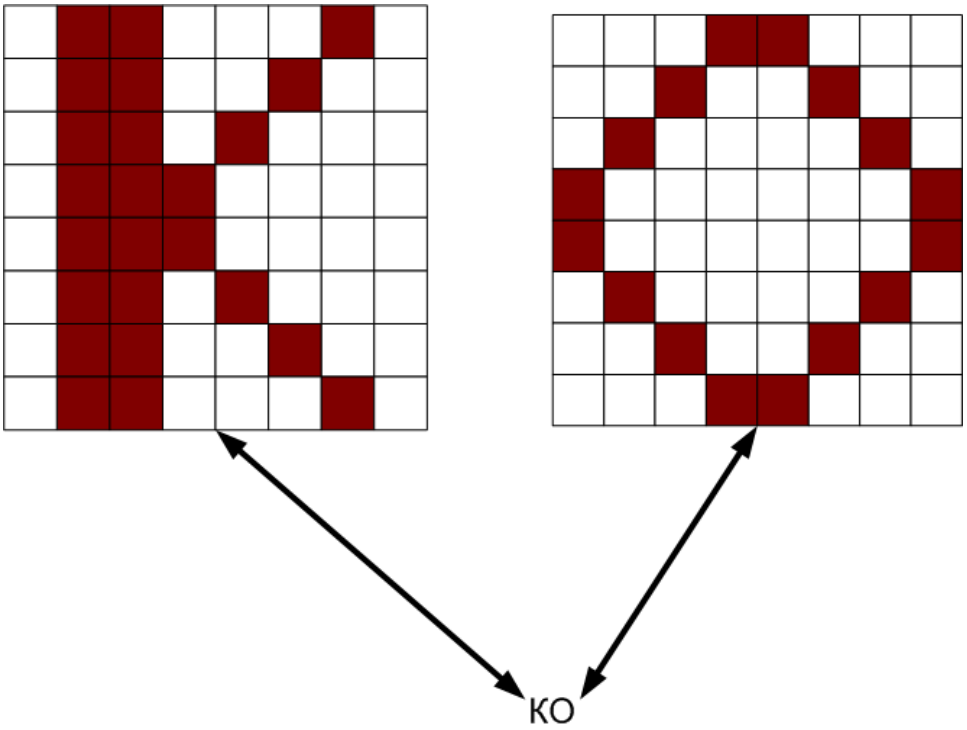



Рисунок 20. Пример растрового шрифта

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 ANSI языка Си определены следующие базовые типы переменных:

| Тип | Описание |
|---|---|
| int | Знаковый целый |
| char | Символьный |
| float | Вещественный одинарной точности с плавающей запятой |
| double | Вещественный двойной точности 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 (signed или unsigned), увеличенный или сокращённый диапазон значений (long, long long (начиная со стандарта C99) или short).

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) целое десятичное число со знаком;

%o – целое без знака в восьмеричной системе счисления;

%x – целое без знака 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 неустановленным – 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, 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-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 **поразрядные 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, результат выполнения операции 2 & 5 будет равен 0 (т.к. 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 младшем разряде, после чего будет произведена операция поразрядного И с единицей (т.е. все разряды числа, кроме младшего, будут умножены на 0). В переменную flag будет возвращена 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 нуль, 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 нуль 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).

## Контрольные вопросы

1. Что такое система счисления? Назовите отличия позиционных систем счисления от непозиционных. Приведите примеры систем счисления обоих видов.
2. Как перевести число из двоичной системы счисления в десятичную и наоборот? Приведите примеры.
3. Как перевести число из восьмеричной системы счисления в десятичную и наоборот? Приведите примеры.
4. Как перевести число из шестнадцатеричной системы счисления в десятичную и наоборот? Приведите примеры.
5. Как перевести число из шестнадцатеричной системы в двоичную и наоборот? Приведите примеры.
6. Как перевести число из восьмеричной системы в двоичную и наоборот? Приведите примеры.

7. Как перевести число из шестнадцатеричной системы в восьмеричную и наоборот? Приведите примеры.
8. Как представляются отрицательные числа в ЭВМ? Что представляет собой дополнительный код? Приведите примеры перевода отрицательных десятичный чисел в двоичную систему счисления.
9. Расскажите о представлении вещественных чисел в ЭВМ. В чем отличие представления таких чисел в форме с фиксированной и плавающей запятой?
10. Как представляется символьная информация в ЭВМ? Какие виды кодировок символов Вы знаете?
11. Объясните отличия стандарта ASCII и ISO 8859-X.
12. Какие кодовые страницы используются в операционных системах компании Microsoft?
13. Что такое шрифт? Какие виды шрифтов Вы знаете?
14. Какие типы данных используются в языке программирования Си для хранения переменных?
15. Что такое двоичный флаг? Каково его назначение?
16. Какие поразрядные операции существуют в языке программирования Си?
17. Что такое маска и маскирование?

# 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 присоединённой к специальному процессору (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.

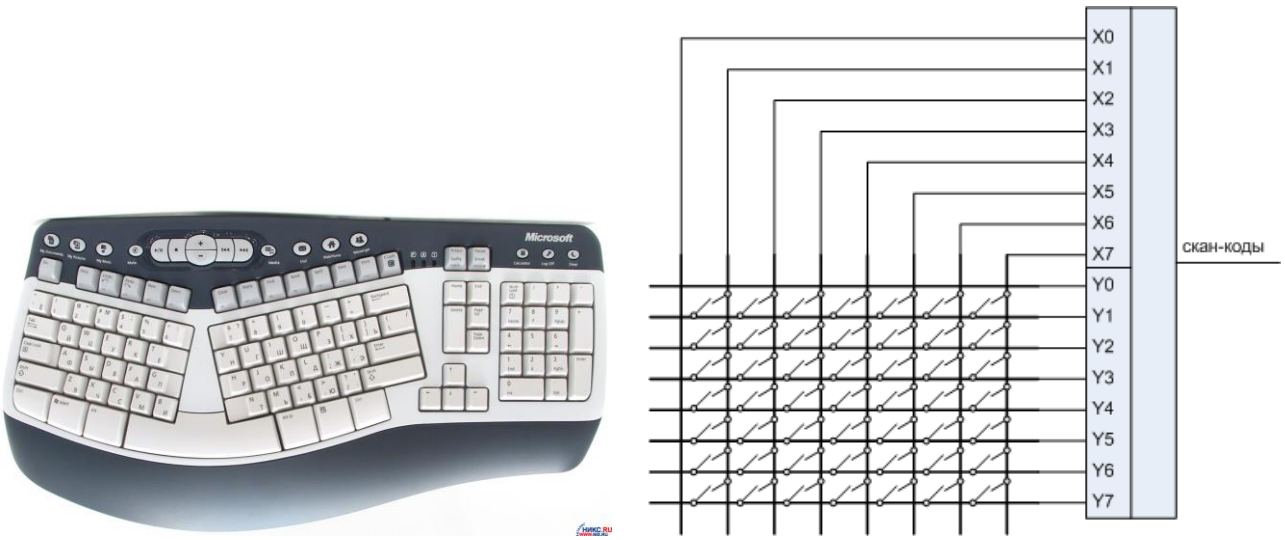



Рисунок 21. Пример клавиатуры (слева) и её простейшая схема (справа)

Скан-код клавиши может состоять из одного или нескольких байтов (см. приложение). Обычно клавиши, на которых нанесены символьные обозначения, имеют однобайтовые коды, а управляющие клавиши – многобайтовые. 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», «а», «А».

**Обратите внимание (!!!)**, что скан-код клавиши и код символа – это разные вещи. Одному скан-коду в зависимости от режима работы клавиатуры мо-

жет соответствовать несколько символов или вообще может не соответствовать никакого символа, как, например, у управляющих клавиш.

Если пользователь продолжает держать клавишу нажатой и, соответственно, переключатель 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 также 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 нанесён символ) или управляющей (например, клавиша перемещения курсора) и т.п.

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

Для некоторых клавиш вообще не формируются 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 чём 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 используются 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 надёжны, их контакты обычно самоочищающиеся. Они выдерживают до 20 миллионов срабатываний и стоят вторыми по долговечности после ёмкостных датчиков (см. ниже).

Клавиши с замыкающими накладками (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 её 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.

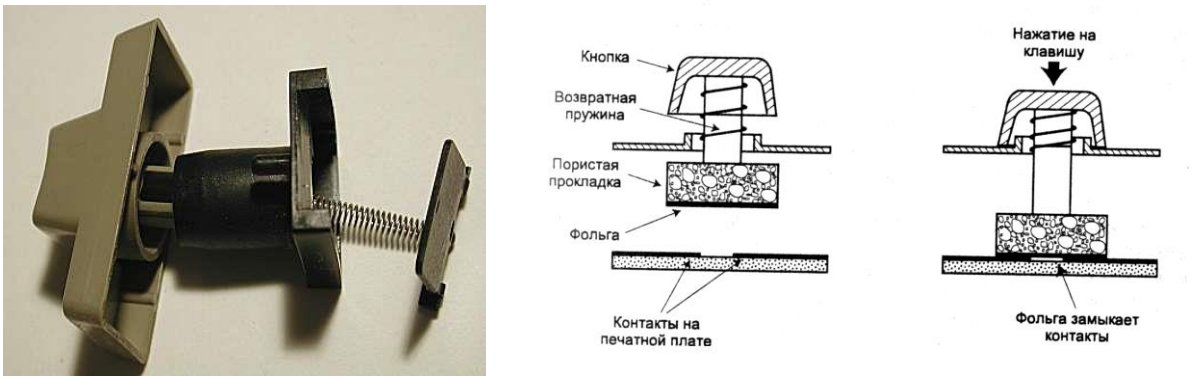



Рисунок 22. Клавиши с замыкающимися накладками

Клавиатура с резиновыми колпачками (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 её от пыли, грязи и влаги.

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.

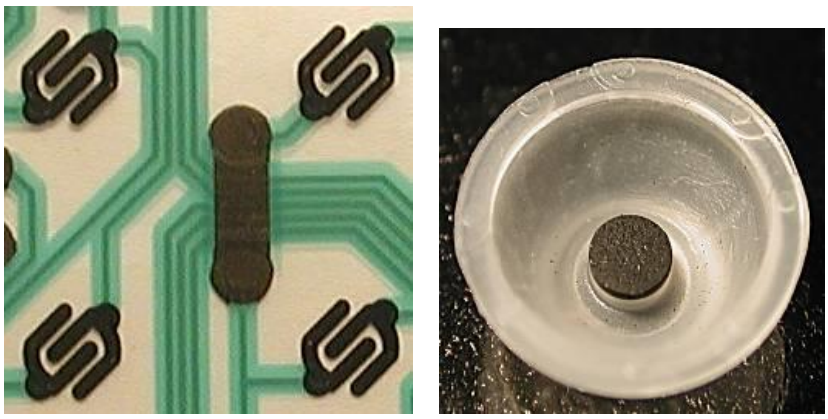


Рисунок 23. Клавиши с резиновыми колпачками

Ёмкостные датчики (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 ёмкости 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, ёмкость между ними увеличивается, что регистрируется схемой компаратора, установленной в клавиатуре.

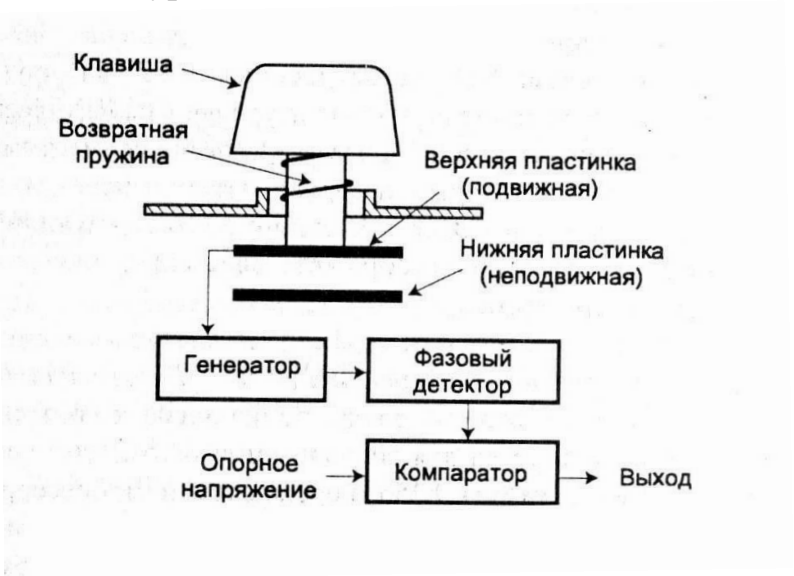



Рисунок 24. Устройство ёмкостной клавиши

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). Её долговечность составляет до 25 миллионов срабатываний. Единственным недостатком такой клавиатуры является её высокая стоимость, но она во многом компенсируется удобством и долговечностью.

## 5.2. Монитор

Монитор, также называемый дисплеем, – это устройство, предназначенное для отображения информации. По физическому принципу получения изображения мониторы можно разделить на: электронно-лучевые (ЭЛТ, англ. Cathode Ray Tube, CRT) и жидкокристаллические (ЖКД, англ. Liquid Crystal Display, LCD).

В первом случае картинка выводится с использованием стеклянной вакуумной электронно-лучевой трубки (рис. 25).


Рисунок 25. Монитор с электронно-лучевой трубкой

С фронтальной стороны внутренняя часть стекла трубки покрыта специальным веществом – люминофором, испускающим свет при попадании на него заряжённых 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 трёх пушек соответствует одному из основных цветов и посылает пучок электронов на различные люминофорные частицы, чьё свечение основными цветами с различной интенсивностью комбинируется, и в результате формируется изображение с требуемым цветом.

Наборы точек люминофора располагаются по треугольным триадам. Триада образует пиксель – 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 развёртки**. А частота, с которой меняются кадры изображения, называется **кадровой частотой развёртки**.

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

Жидкокристаллический монитор представляет собой две стеклянных или пластиковых пластины, между которыми находится суспензия (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.

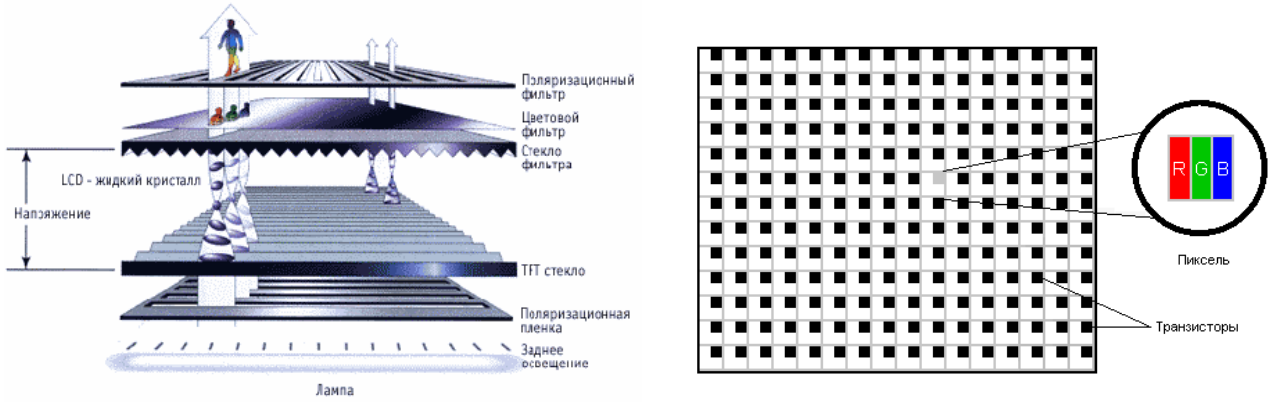



Рисунок 26. Монитор с жидкокристаллическим экраном

Существует два вида жидкокристаллических мониторов: DSTN (англ. Dual-Scan Twisted Nematic – кристаллические экраны с двойным сканированием) и TFT (англ. Thin Film Transistor – на тонкоплёночных транзисторах), также их называют соответственно пассивными и активными матрицами. Такие мониторы состоят из следующих слоёв: 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 – пикселей. Как и в электроннолучевых трубках, пиксель формируется из трёх участков – красного, зелёного 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 (рис. 27).

Рисунок 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, цифро-аналогового преобразователя (ЦАП, англ. Digital-to-Analog Converter, DAC) и постоянного запоминающего устройства (ПЗУ).

Память служит непосредственно для хранения изображения, представляемого в виде набора точек – экранных пикселей. От объёма 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 ЦАП.

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, курсора, 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 (рис. 28): D-sub и DVI. Первый используется для подключения аналоговых мониторов (которые получают сигнал в аналоговой форме), второй – для подключения цифровых мониторов (получающих сигнал в цифровой форме).

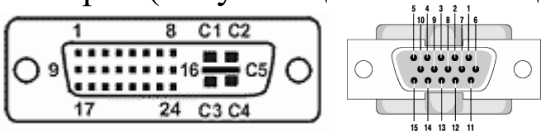



Рисунок 28. Разъёмы для подключения мониторов (DVI – слева, D-sub – справа)

### 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, соединённое с ЭВМ (или каким-то другим устройством) через кабельные или телефонные каналы (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.

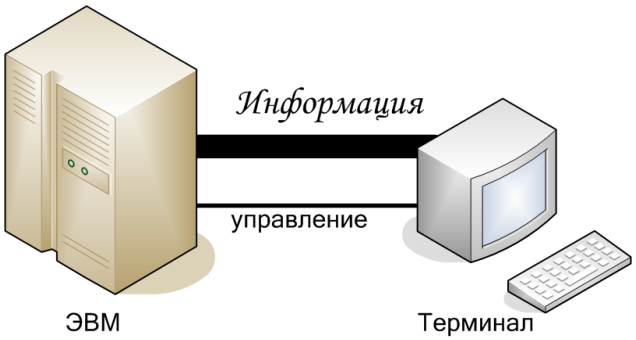



Рисунок 29. Использование терминала для доступа к ЭВМ

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 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 (рис. 30).

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

управляющие последовательности), и поступает на вход 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 сразу дублироваться на экране терминала. Такой режим называется «ЭХО».

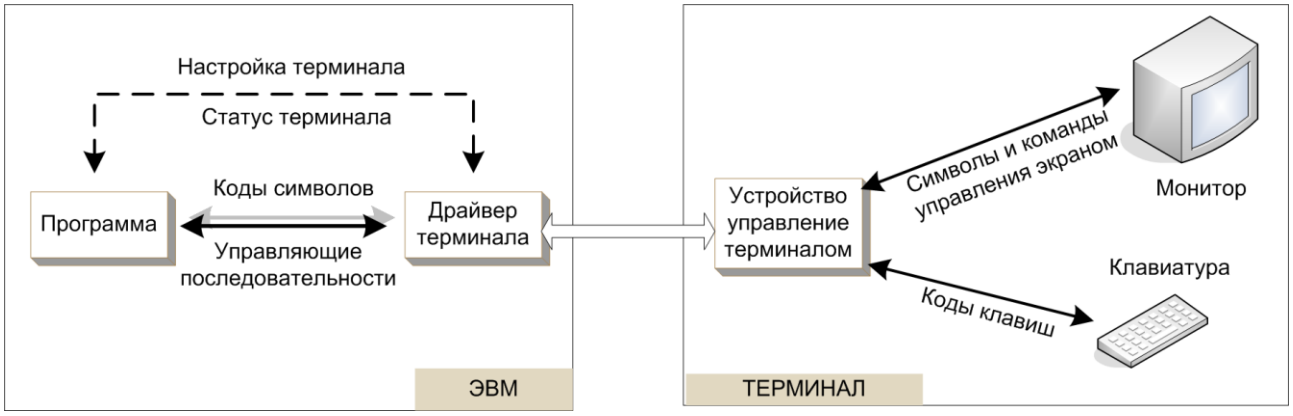



Рисунок 30. Принцип работы терминала

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 передаётся 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 его текущие настройки. Программа пользователя также может управлять терминалом (например, передвинуть курсор на экране или изменить цвет выводимых символов) путём передачи ему специальных последовательностей 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 коды таких символов: 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. Эти символы обычно не показываются на экране.

Так как не имеется достаточного количества управляющих кодов, чтобы делать всё, используются команды терминала, чаще всего называемые **Escape-**

**последовательностями**. Они состоят из нескольких подряд идущих символов, первым из которых является символ с кодом ASCII, равным 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 и выполнить соответствующую команду. Когда распознаётся 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 подключённые 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 служат 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, поочерёдно 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\30
4q\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 6 символов (\E, [, H, \E, [, J), то произойдёт очистка экрана, и курсор переместится в левый верхний угол экрана. Интерес вызывает команда set_a_background (установить фон), значение которой равно: \E[4%p1%dm, что означает, что команда должна формироваться с использованием одного параметра (%p1), имеющего целый тип (%d) и располагающегося 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 являются правила вы-

вода символов псевдографики (символов, позволяющих выводить текстовые рамки). Суть проблемы заключается 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 послать ему управляющую команду, указанную в поле enter_alt_charser_mode. Команда, необходимая для обратного переключения, хранится в поле exit_alt_charset_mode.

Чтобы определить место расположения символов псевдографики, используют строку соответствия символов (поле acs_chars) тому расположению, которое 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 команде terminfo.

### 5.5. Взаимодействие с терминалом в ОС Linux

Для взаимодействия пользователей с ПК, работающим под управлением операционной системы 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, подключённым к ЭВМ.

В ОС Linux пользователи «изолируются» от аппаратурной 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. Причём 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 жёсткие диски;
- символьные устройства, т.е. использующие побайтную передачу данных. Терминалы относятся именно к таким устройствам.

Заметим, что это разделение условно, так как одно и то же 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 «жёсткий 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 применяется системный вызов *ioctl*. Для работы с терминалами – дополнительные функции *isatty*, *tcgetattr*, *tcsetattr*, позволяющие произвести настройку драйвера на необходимый режим работы и определить его текущее 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 передаются строковая константа, соответствующая имени открываемого файла, режим 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 передаётся в качестве аргумента для функций, работающих с устройством.

Параметр *flags* определяет, в каком режиме требуется открыть файл. Он является целым числом, в котором за каждым битом однозначно закреплён один требуемый режим. Для наглядности определения требуемых режимов в заголовочном файле *fcntl.h* библиотеки функций работы с файлами заданы соответствующие макросы. Например, режим открытия для чтения 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 файл с указанным дескриптором.

**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 0, 1 и 2, соответственно. 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 файла, 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 fd);
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 терминалы автоматически открыты для потоков ввода, вывода и ошибок при её выполнении, 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 передаётся строка «Поток ввода связан с терминалом» и выводится полное имя этого файла (строка 7). В противном случае в поток вывода передаётся 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). Эта функция возвращает число прочитанных байтов или –1 в случае ошибки.

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.

**Описание функции *read* и *write***

```
#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 записывающую 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.п. Для демонстрации работы программы её необходимо запустить на любом текстовом терминале, а на втором текстовом терминале зайти под своим учётным именем (так как программа попытается вывести информацию на 2-й терминал, а сделать она это сможет только, 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 терминала 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)     return (0);
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 опера-

ции. 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 команд, которые она может выполнять 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 четырёх регистров флагов, описывающих работу драйвера терминала при обработке входной информации (от клавиатуры), при выводе информации на экран, при передаче информации в ЭВМ, дополнительных режимов, а также массива 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 важ-

ные 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 неканонический). 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 (англ. 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 запроса на чтение данных в неканоническом режиме. 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 символов, 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 приходе первого же символа или по истечении 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.

**Описание функций *tcgetattr* и *tcsetattr***

```
#include <termios.h>
#include <unistd.h>

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

Для установки новых параметров драйвера терминала используется вызов *tcsetattr*, который 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 ждёт опустошения очереди вывода.
- **TCSAFLUSH**. Аналогично TCSADRAIN ждёт, пока очередь вывода не опустеет, а затем также очищает и очередь ввода перед установкой для параметров дисциплины линии связи значений, заданных в структуре *tnew*.

## Контрольные вопросы

1. Для чего используется клавиатура? Что такое скан-код? Расскажите, как ЭВМ узнаёт, какая клавиша была нажата.
2. Что представляет собой код клавиши? Какие типы клавиш Вы знаете?
3. Для чего предназначен монитор? На какие типы можно разделить мониторы по физическому принципу получения изображения?
4. Расскажите об устройстве электронно-лучевых мониторов. Что такое строчная и кадровая частота развёртки?
5. Расскажите об устройстве жидкокристаллического монитора.
6. Для чего используется видеоадаптер? Расскажите о структуре и составе видеоадаптера.
7. Что такое терминал? Приведите примеры терминалов. Расскажите о принципе работы терминала.
8. Какие режимы ввода информации через терминал существуют?
9. Как может управлять терминалом программа пользователя?
10. Что такое Escape-последовательность? Как получить список доступных команд для терминала?
11. Как перейти от основной таблицы кодировок символов в терминале к дополнительной?
12. Какие устройства относятся к блочным, а какие – к символьным?
13. Какие функции используются для прямого доступа к файлам?
14. Для чего используются функции *isatty*, *ttyname*?
15. Какие функции используются для управления терминалом?
16. В какой структуре описаны параметры работы терминалов? Объясните назначение полей этой структуры.
17. Для чего используются функции *tcgetattr* и *tcsetattr*?

# 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 происходящее в ЭВМ событие, 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 аппаратурно-программными средствами. Для организации аппаратурной системы прерываний используется контроллер прерываний, который подключается к соответствующему входу микропроцессора. К нему в свою очередь подключаются внешние устройства (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 аппаратурных, 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 и обычные программы. После её завершения автоматически загружается ранее прерванная программа, и продолжается её выполнение.

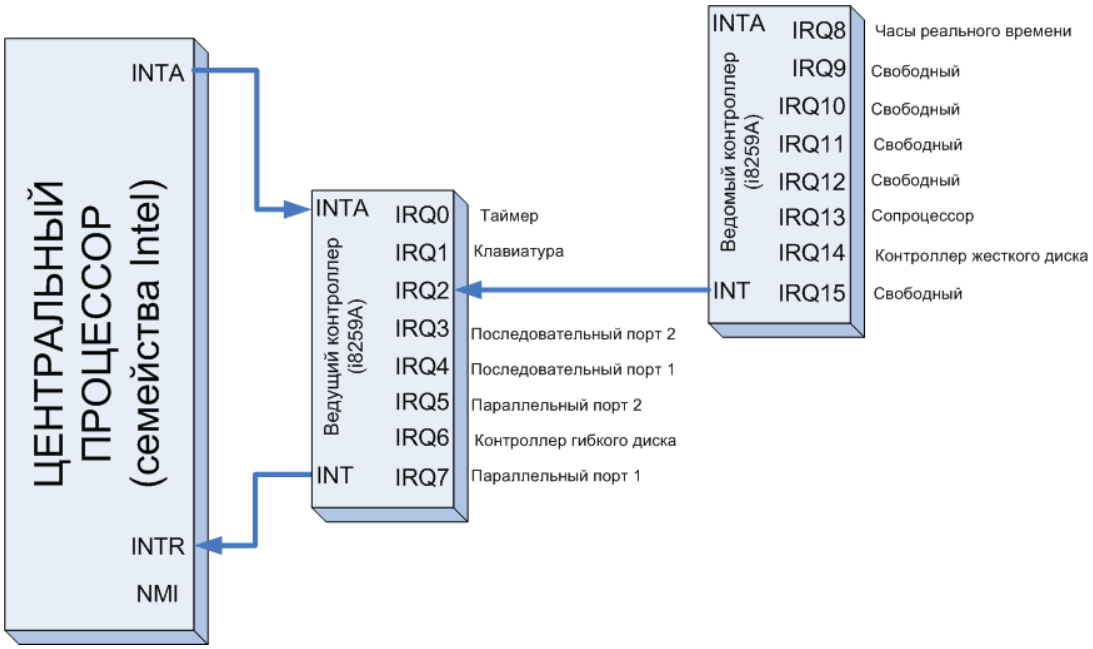



Рисунок 31. Схема каскадного подключения контроллеров прерываний в ПК на базе процессоров семейства Intel

## 6.2. Обработка программных прерываний в UNIX-подобных 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, о появлении в очереди управляющих символов или возникновении ошибки во время работы программы (например, Segmentation Fault – 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). Перечислим некоторые из них:

- **SIGABRT** – сигнал прерывания процесса (англ. process abort signal). Посылается процессу при вызове им функции *abort()*. В результате сигнала SIGABRT произойдет аварийное завершение

(англ. abnormal termination) и запись образа памяти (англ. core dump, иногда переводится как «дамп памяти»). Образ памяти процесса сохраняется в файле на диске для изучения с помощью отладчика;

- **SIGALRM** – сигнал таймера (англ. alarm clock). Посылается процессу ядром при срабатывании таймера. Каждый процесс может устанавливать не менее трёх таймеров. Первый из них измеряет прошедшее реальное время. Этот таймер устанавливается самим процессом при помощи системных вызовов *alarm* или *setitimer* (см. ниже);
- **SIGILL** – недопустимая команда процессора (англ. 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 (англ. interrupt). Посылается ядром всем процессам, связанным с терминалом, когда пользователь нажимает клавишу прерывания (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 (англ. 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 определённой пользователем процедуры);
- **SIGPROF** – сигнал профилирующего таймера (англ. profiling time expired). Как было уже упомянуто для сигнала SIGALRM, любой процесс может установить не менее трёх 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 (англ. quit). Очень похожий на сигнал SIGINT. Этот сигнал посылается ядром, когда пользователь нажимает клавишу выхода в используемом терминале. Значение клавиши выхода по умолчанию соответствует символу F6 таблицы ASCII или CTRL+Q. В отличие от SIGINT этот сигнал приводит к аварийному завершению и сбросу образа памяти;

- **SIGSEGV** – обращение к некорректному адресу памяти (англ. invalid memory reference). Сокращение SEGV в названии сигнала означает нарушение границ сегментов памяти (англ. segmentation violation). Сигнал генерируется, если процесс пытается обратиться к неверному адресу памяти. Получение сигнала SIGSEGV приводит к аварийному завершению процесса;
- **SIGTERM** – программный сигнал завершения (англ. 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 (англ. 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 трёх 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 определённое 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 передаётся номер полученного сигнала) или 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 начнёт выполнять указанную функцию. После её завершения выполнение программы, получившей сигнал, будет продолжено, и обработчик сигнала будет установлен в *SIG_DFL*.

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

**Описание функции *raise***

```
#include <signal.h>

int raise (int signum);
```

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

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

**Описание функции *pause***

```
#include <unistd.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* вернёт значение –1 и поместит в переменную *errno* значение EINTR.

Пример программы, обрабатывающей прерывания, приведён в листинге 4.

**Листинг 4. 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 подаёт сигнал SIGALRM, когда значение таймера становится равным 0.
- **ITIMER_VIRTUAL** уменьшается только во время работы процесса и подаёт 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, подаётся сигнал SIGPROF.

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

Для установки таймеров используется функция *setitimer*. Величина, в 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 в нуль (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, const 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, приведён в листинге 5.

**Листинг 5. 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) }
```

## Контрольные вопросы

1. Что такое прерывание? Какие механизмы обработки прерываний Вы знаете?

2. Объясните назначение сигналов. Назовите известные Вам типы сигналов, объясните, для чего они используются.
3. Какая функция используется для определения действия, которое необходимо выполнить при получении сигнала?
4. Как вызвать сигнал из программы?
5. Для чего используется функция *pause*?
6. Что такое таймер? Какие типы таймеров можно устанавливать в программе?
7. Какой сигнал подаёт таймер *ITIMER_REAL*?
8. Какие функции используются для установки таймеров?
9. Расскажите о назначении полей структур *itimerval* и *timeval*.

# ГЛАВА 7. НАКОПИТЕЛИ НА ЖЁСТКИХ 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, жёсткие диски, стримеры (устройства хранения данных на магнитных лентах), ZIP, АРВИД и т.п. Здесь внимание будет уделено основным принципам работы жёстких дисков, так как в современных персональных компьютерах они наиболее часто используются для хранения информации.

*Накопители на жёстких 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 распространённой модели ружья фирмы «Winchester», в результате чего для обозначения дисковых устройств с несъёмными дисками стали широко применять это слово.*

## 7.1. Конструкция дисководов жёстких дисков

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

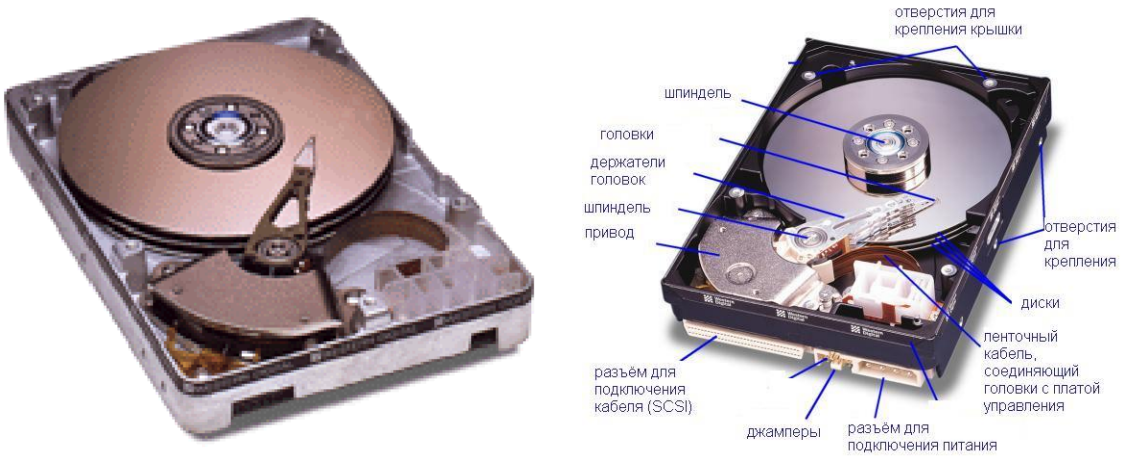



Рисунок 32. Устройство хранения данных на жёстких магнитных дисках (винчестер)

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

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 распространёнными являются два типа слоя:

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

- *тонкоплёночный* – состоящий из нескольких слоёв 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 плёночные 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 нём 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 контроллером жёсткого диска головка перемещается на нужное место, 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 счёт 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 серьёзные ограничения на их исполь-

зование 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 твёрдой 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);
- *тонкоплёночные головки* создаются методом фотолитографии. Магнитный сердечник головки осаждается на керамическую поверхность, что позволяет создать головки с очень малым магнитным зазором. Такая технология даёт 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* (англ. 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 0,1 Н, а при вращении диска между головкой и поверхностью образовывался воздушный зазор толщиной около 0,5 мкм.

В современных устройствах модуль данных и привод составляют единое целое. Система подачи очищенного воздуха уже не используется. Для надёжной 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 жёстком 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.

Жёсткий диск имеет несколько разъёмов:

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

Форм-фактор интерфейсного разъёма определяет стандарт взаимодействия жёсткого 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 жёстких дисках

Вся информация на диске записывается в форме концентрических окружностей, называемых **дорожками** (рис. 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 – сектора, которые являются минимальной единицей информации на жёстком диске.

Сектор состоит из трёх 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 жёсткого 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 объём хранимой информацией. Другими словами, геометрия – это максимальное число цилиндров (C), число головок (H) и число секторов на дорожку (S).

Ёмкость = H×C×S×(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)×100×200×512 = 81920000 байт (или 80 000 КБ). Необходимо помнить, что на один диск приходится по 2 головки, и один КБ равен 1024 байтам.

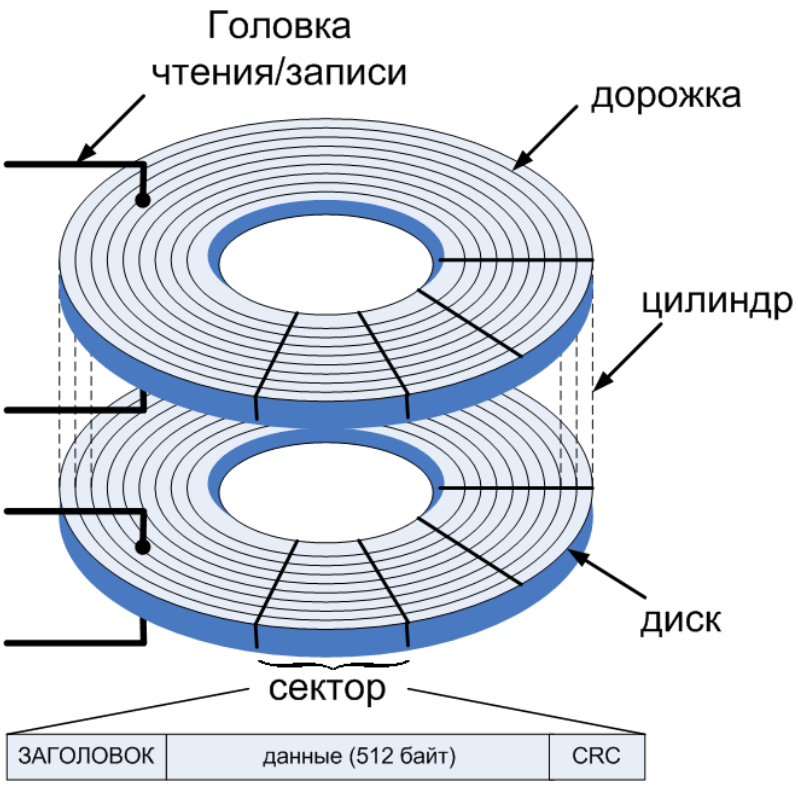



Рисунок 33. Способ хранения информации на жёстком диске

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 ГБ при размере сектора в 512 байт.

Со временем, когда размера диска в 128 ГБ стало не хватать, стали применять **линейную** адресацию секторов (англ. Logical Block Addressing, LBA), в которой все секторы на диске нумеровались целыми числами, начиная с 0. Получая такой адрес, контроллер жёсткого 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 (рис. 34) CHS-адресации и геометрии в LBA и наоборот. Это преобразование осуществляется специальной программой – драйвером, который используется для доступа программ пользователя к информации на жёстком диске.

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

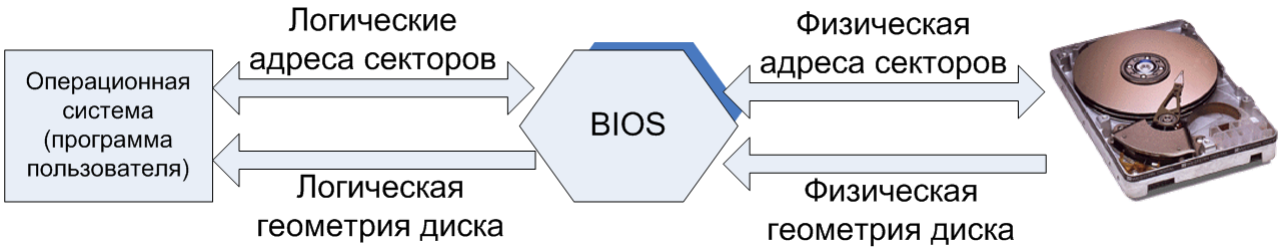



Рисунок 34. Принцип трансляции адресов секторов и геометрий жёстких дисков

### 7.3. Барьеры размеров жёстких дисков. Трансляция адресов

Стремясь сократить объём 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» – 8 бит, а для «S» – 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-ми младшим разрядам номера цилиндра.

Таким образом, получилось **ограничение BIOS** (или DOS, которая использовала только драйвер BIOS) **в 8,5 ГБ**. Сегодня это является серьёзным ограничением размера диска. 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 жёсткого 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 МБ, первоначально стали использовать приём (сейчас в современных BIOS он называется Large) уменьшения числа цилиндров (кратное степени 2) с одновременным увеличением числа головок. Таким образом, появилась возможность использовать диски объёмом до 8,4 ГБ, но с «фальшивой» (логической) геометрией. При чтении информации с жёсткого 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 жёсткого диска будет происходить потеря размера диска (за счёт округлений значений чисел до целых).

Например, пусть имеется винчестер со следующей физической геометрией: C = 1238, H = 16, S = 63 (объём = 638 МБ), логическая адресация будет следующей: C = 619, H = 32, S = 63 (объём = 638 МБ). Теперь всё программное обеспечение, использующее функцию BIOS для доступа к диску, считает, что работает с жёстким диском, в котором 32 головки вместо 16 и 619 цилиндров вместо 1238.

Перевод логического CHS-адреса в физический CHS (и обратно) осуществляется следующим образом (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 $C^Г_л$ – число цилиндров в логической геометрии, $H_ф$ – число головок в физической геометрии, $K$ – делитель, полученный при преобразовании физической геометрии в логическую геометрию. Запись [$X$] означает целочисленное деление. Запись $X$%$Y$ означает получение остатка от целочисленного деления $X$ на $Y$.


Рисунок 35. Преобразование CHS-адресов при использовании Large трансляции

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

В этом случае для получения геометрии BIOS выполняются следующие действия. Считается, что на диске все дорожки всегда делятся на 63 сектора. Номер максимального сектора делится на 63 (логическое число секторов). Полученное число ($L_1$) 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 ($H_л$). Остаток от деления $L_1$ на $H_л$ будет число цилиндров в логической геометрии ($C_л$).

Например, имеется жёсткий 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 CHS-геометрия принимает вид – 623/255/63 (т.е. считается, что имеется 623 цилиндра (0-622), 255 головок (0-254), 63 сектора (1-63)).

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

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

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

где $S_{LBA}$ – номер максимально возможного сектора в LBA геометрии. $S^Г_л$ – число секторов в логической геометрии.


Рисунок 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 жёсткого диска взаимодействует с BIOS стандартным образом, и сам преобразует получаемые адреса.

### 7.4. Логическая организация винчестера. Таблица разделов

В современных персональных компьютерах жёсткие 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 жёсткий диск, хранится в виде **таблиц разделов**, располагаемых в специально отведённых секторах. Обычно для формирования этих таблиц используются служебные программы, например FDISK.

При подготовке любого жёсткого диска в его первый сектор (цилиндр 0, головка 0, сектор 1) заносится информация, называемая **главной загрузочной записью** (англ. Master Boot Record, MBR). В ней содержится **программа-загрузчик ядра операционной системы** (446 байт), основная **таблица разделов** (64 байта), с которой начинается поиск информации и разбиение жёсткого 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 таблице 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 ГБ.

Первым байтом в структуре, описывающей раздел, идёт флаг активности раздела (0 – не активен, 128 (80H) – активен). Он служит для определения, является ли раздел системным загрузочным и необходимо ли производить загруз-

ку 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-й 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. Значения кодов приведены в таблице 4.

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

Далее идёт адрес начального сектора в формате LBA и размер раздела в секторах.

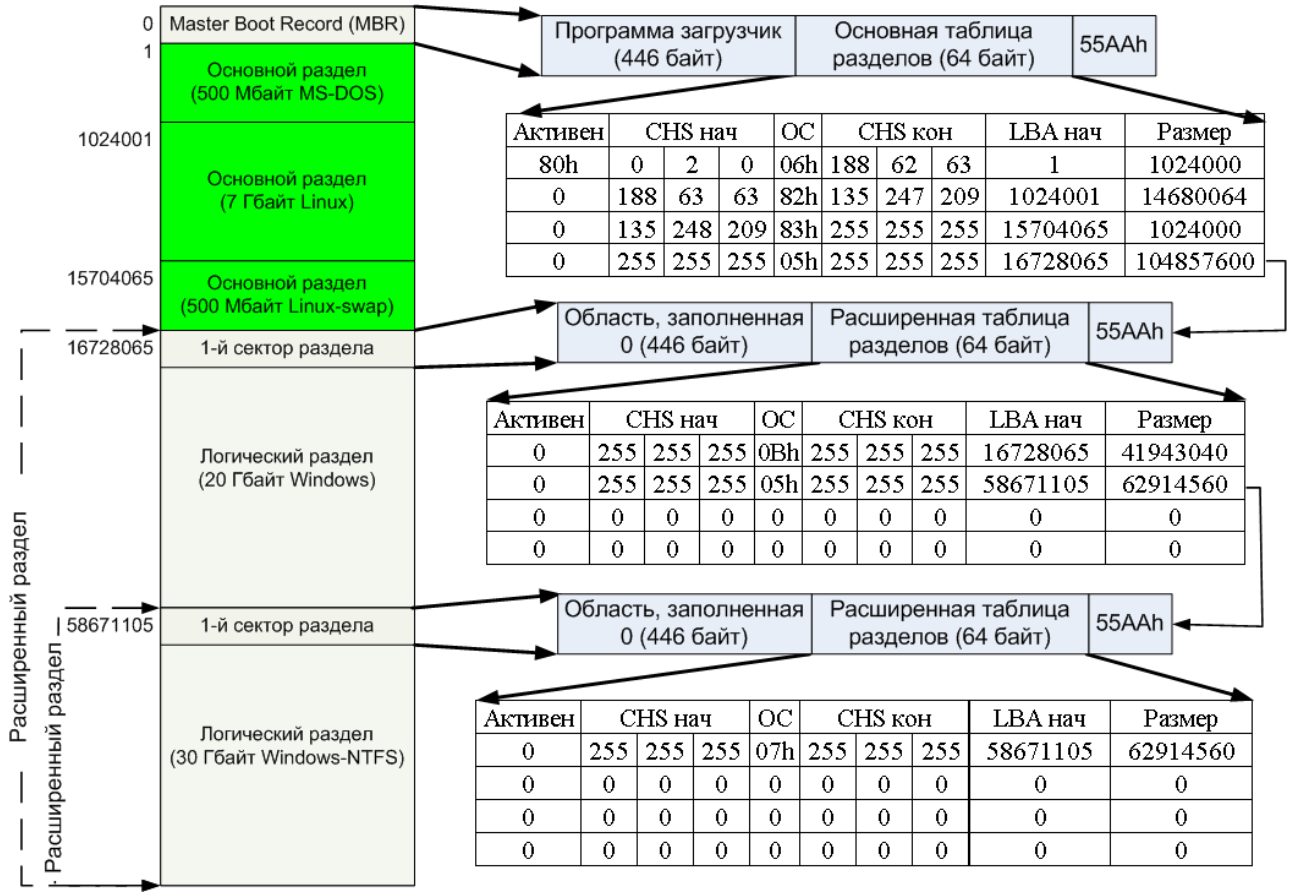
Таблица 4. Типы разделов жёсткого диска

| 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 (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 | | | ОС | 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 | | | ОС | 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 | | | ОС | 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 |

Рисунок 37. Пример полной таблицы разделов для жёсткого диска размером 60 ГБ

## Контрольные вопросы

1. Что представляет собой жёсткий диск? Расскажите об основных элементах жёсткого диска.
2. Каково назначение магнитных головок? Зачем нужна парковка головок?
3. Что такое дорожка, цилиндр, сектор? Что называется геометрией жёсткого диска?
4. Как происходит адресация данных на жёстких дисках?
5. В чём недостатки CHS-адресации?
6. Как организована LBA-адресация?
7. Расскажите об отличиях логической и физической геометрий дисков? Зачем была введена логическая адресация?

8. Как производится перевод логического CHS-адреса в физический CHS-адрес и обратно?
9. Как производится перевод CHS-адресов в LBA и обратно?
10. Где хранится информация о разделах жёсткого диска? Что такое раздел?
11. Расскажите о структуре таблицы разделов.
12. Перечислите типы разделов жёсткого диска.

# ГЛАВА 8. БАЗОВЫЕ ЭЛЕМЕНТЫ ЭВМ. АЗЫ БУЛЕВОЙ АЛГЕБРЫ

Основными элементами, на которых строятся ЭВМ, являются простые цифровые устройства – вентили, которые могут принимать два значения – «нуль» и «единица». Чаще всего состоянию «единица» соответствует положительное напряжение (например, + 5 В) на его выходе, состоянию «нуль» – 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 изображён 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 в противоположное. Для переключения транзистора из одного состояния в другое требуется некоторое время, например, несколько наносекунд.

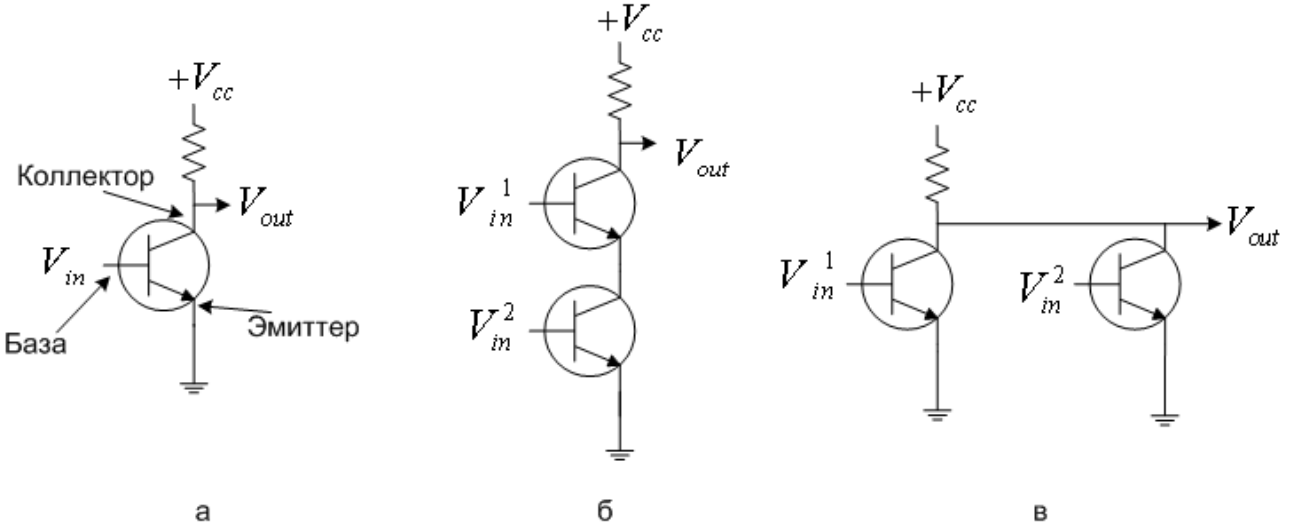



Рисунок 38. Схемы вентилей НЕ (а), И-НЕ (б), ИЛИ-НЕ (в)

Если два транзистора соединить последовательно (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 $V^1_{in}$ и $V^2_{in}$ высокие и соответствуют логическим единицам.

Если два транзистора соединить параллельно (рис. 38в), то схема начинает выполнять функции логического умножения с инверсией. Если напряжения $V^1_{in}$ и $V^2_{in}$ высокие, то оба транзистора открыты, и напряжение $V_{out}$ стремится к 0. Если же хотя бы одно из напряжений $V^1_{in}$ или $V^2_{in}$ низкое, то соответствую-

щий 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 подаётся единица. На выходе второй схемы получится единица, если хотя бы на один из входов подаётся единица.

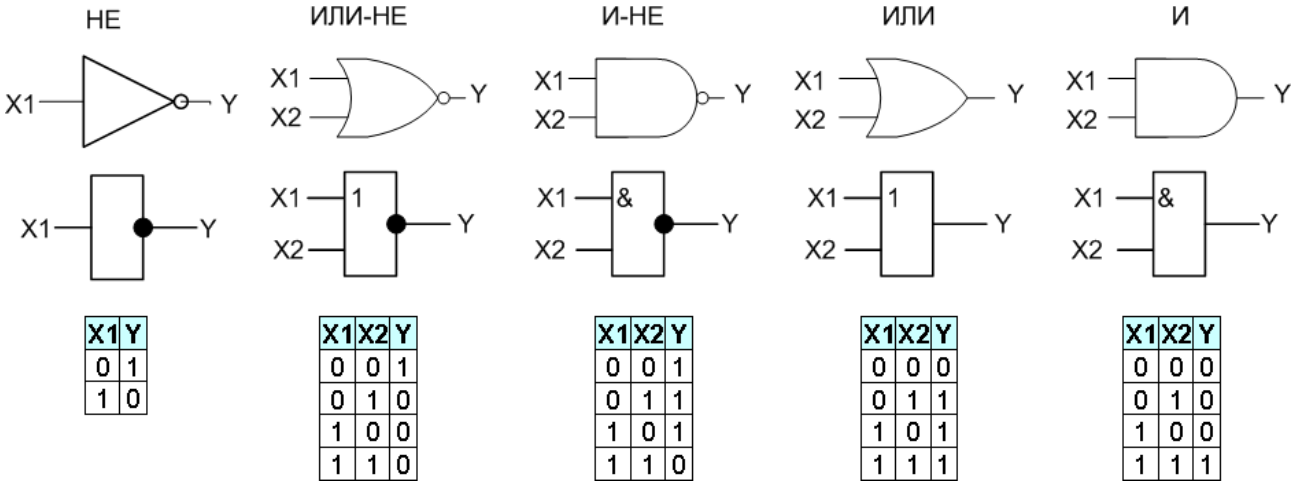
Из выше сказанного ясно, что для реализации схем И-НЕ и ИЛИ-НЕ требуется всего два транзистора, а для схем И и ИЛИ – три. По этой причине чаще всего используются только вентили НЕ, И-НЕ и ИЛИ-НЕ.

## 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 нескольких 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 таблицы истинности и с помощью алгебраической записи.




| 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 |

Рисунок 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)

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).

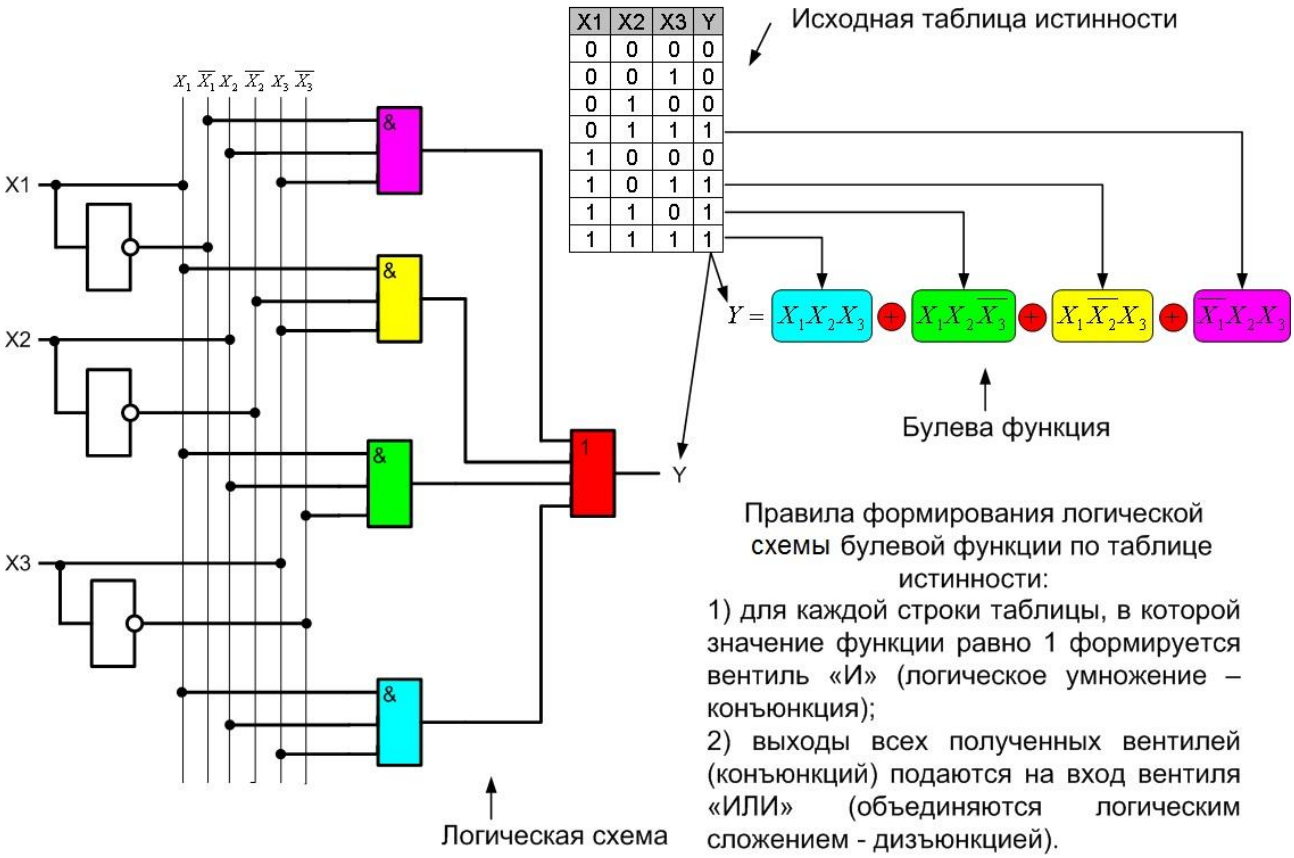



Рисунок 40. Формирование булевой функции и синтез логических схем

Как альтернатива таблицы истинности используется **алгебраическая запись**, в которой перечисляются все комбинации переменных, дающие единичное (или нулевое) значение функции. При этом знаком умножения обозначается операция И, а знаком сложения – 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 рисунке 39. Для отличия вентилей И-НЕ и ИЛИ-НЕ от И и ИЛИ на схемах первых выход обозначается кругом.

Синтез логической схемы осуществляется аналогично получению булевой функции из таблицы истинности (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.

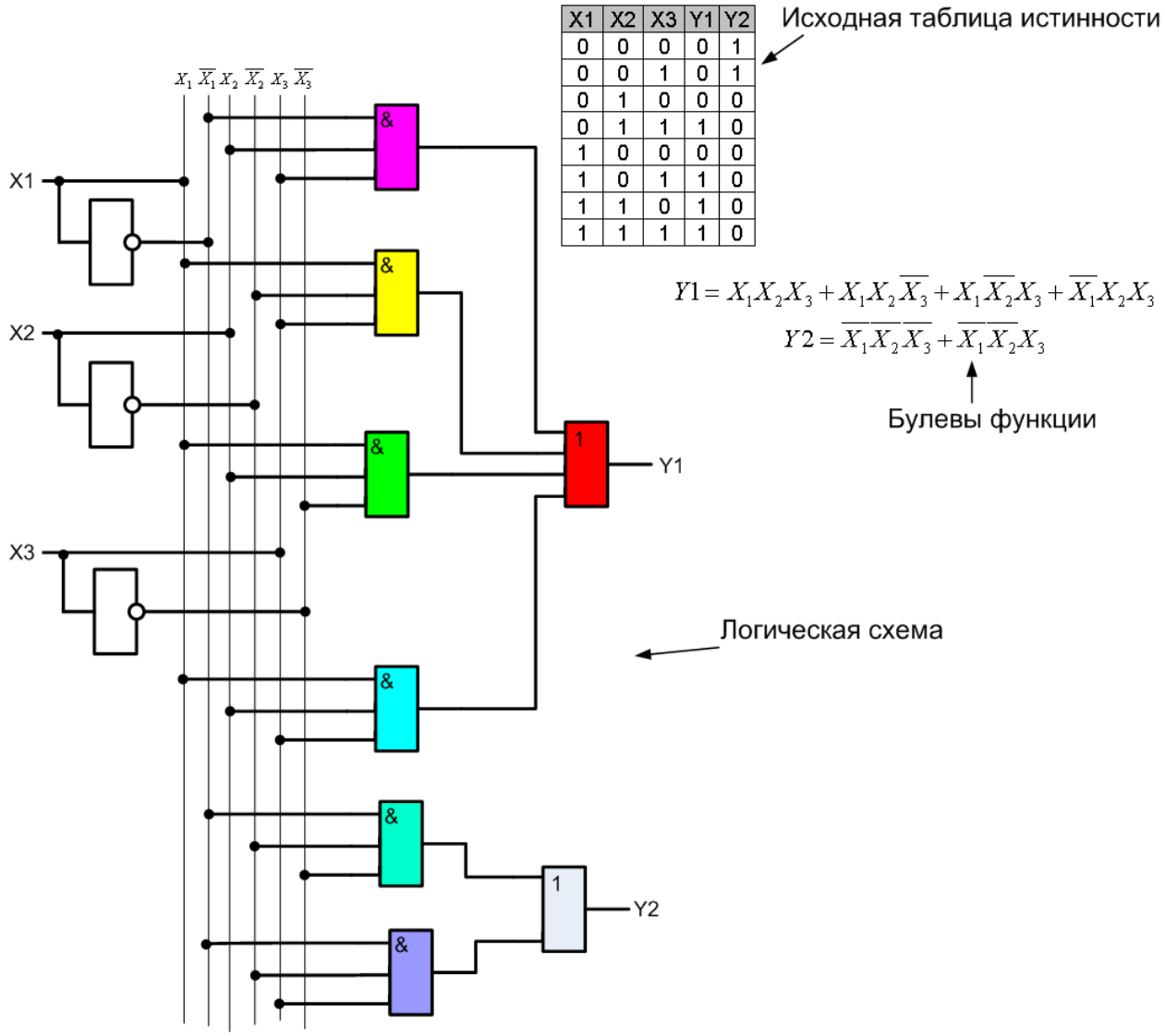



Рисунок 41. Синтез схемы для устройства, имеющего два выхода

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 данного пособия и поэтому рассматриваться не будут.

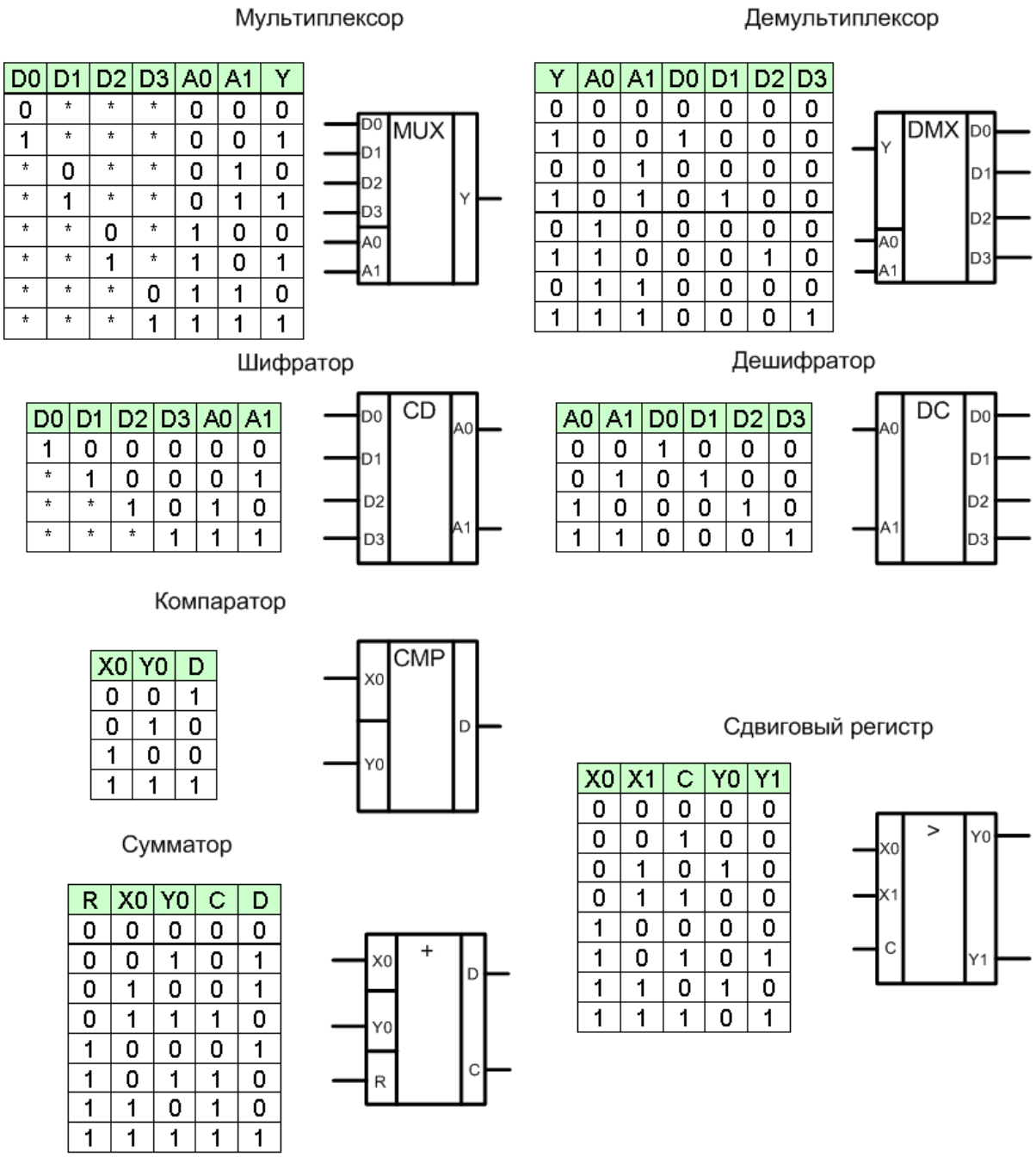



Рисунок 42. Комбинационные схемы, их таблицы истинности и обозначение на логических схемах (символ * в таблице истинности означает любое значение)

### *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 которые выдаётся 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 вправо, если на управляющий вход подаётся единица, и сдвинутому влево, если на управляющий вход подаётся нуль. 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 этого, предусматривается выход, сигнализирующий о переполнении, и вход, учитывающий перенос с предыдущего разряда (R).

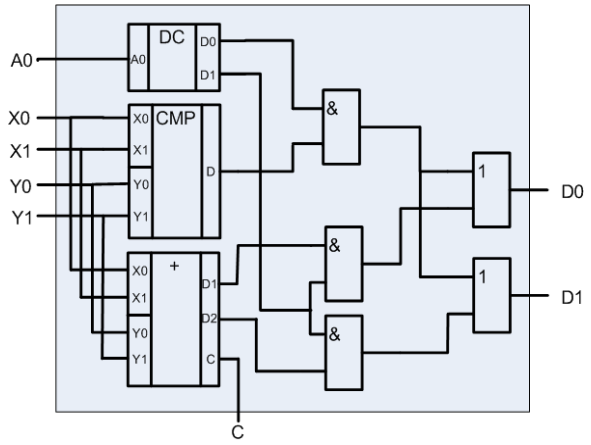



Рисунок 43. Пример схемы АЛУ, выполняющего два действия

Таблицы истинности для рассмотренных комбинационных схем приведены на рисунке 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 подаётся на вход АЛУ, будет выбрана соответствующая часть схемы (используя дешифратор и схему И) и выполнено требуемое действие. Например, на рисун-

ке 43 изображено АЛУ, выполняющее два действия – сложение и сравнение двух чисел.

## Контрольные вопросы

1. Что такое вентиль? Нарисуйте схемы вентилей НЕ, И-НЕ и ИЛИ-НЕ.
2. Что такое булева функция? Назовите виды представления булевых функций.
3. Напишите таблицы истинности для элементов НЕ, И-НЕ и ИЛИ-НЕ.
4. Запишите булевы функции в алгебраическом виде для тех же элементов.
5. Как осуществляется синтез логической схемы? Объясните на примере.
6. Что такое комбинационная логическая схема?
7. Запишите таблицу истинности для мультиплексора и демультиплексора. Объясните назначение этих устройств.
8. Запишите таблицу истинности для шифратора и дешифратора. Объясните назначение этих устройств.
9. Запишите таблицу истинности для сдвигового регистра. Объясните назначение этого устройства.
10. Что такое АЛУ? Нарисуйте схему АЛУ, выполняющего действия сравнения и сдвига.

# СПИСОК ЛИТЕРАТУРЫ


1. Юров В.И. Ассемблер: Учебник для вузов. 2-е изд. – СПб.: Питер, 2011. – 640с.
2. Кутузов М.А., Преображенский А. Выбор и модернизация компьютера. 4-е изд. – СПб.: Питер, 2005. – 320с.
3. Мюллер С., Зекер К. Модернизация и ремонт ПК. 10-е изд.: Пер. с англ. – К.; М.; СПб.: Вильямс, 1999 . – 992с.
4. Томпсон Р.Б., Томпсон Б.Ф. Железо ПК: Энциклопедия. 3-е изд. – СПб.: Питер, 2004. – 956с.
5. Хамахер К., Вранешич З., Заки С. Организация ЭВМ. 5-е изд. – СПб.: Питер, 2003. – 848с.
6. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: Учебник для вузов. – СПб.: Питер, 2007. – 672с.
7. Таненбаум Э. Архитектура компьютера. 5-е изд. – СПб.: Питер, 2007. – 844с.

# ПРИЛОЖЕНИЕ 1. ТАБЛИЦА СКАН-КОДОВ

## КОДЫ КЛАВИШ ДЛЯ 101-,102- и 104-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

В рамках выполнения лабораторных работ и курсового проектирования необходимо разработать программную модель простейшей вычислительной машины Simple Computer. Архитектура Simple Computer представлена ниже.

Для управления моделью (определения начальных состояний узлов Simple Computer, запуска программ на выполнение, отражения хода выполнения программ) требуется создать консоль (рис. П2.1). Необходимо реализовать трансляторы с языков Simple Assembler и Simple Basic для программирования Simple Computer.

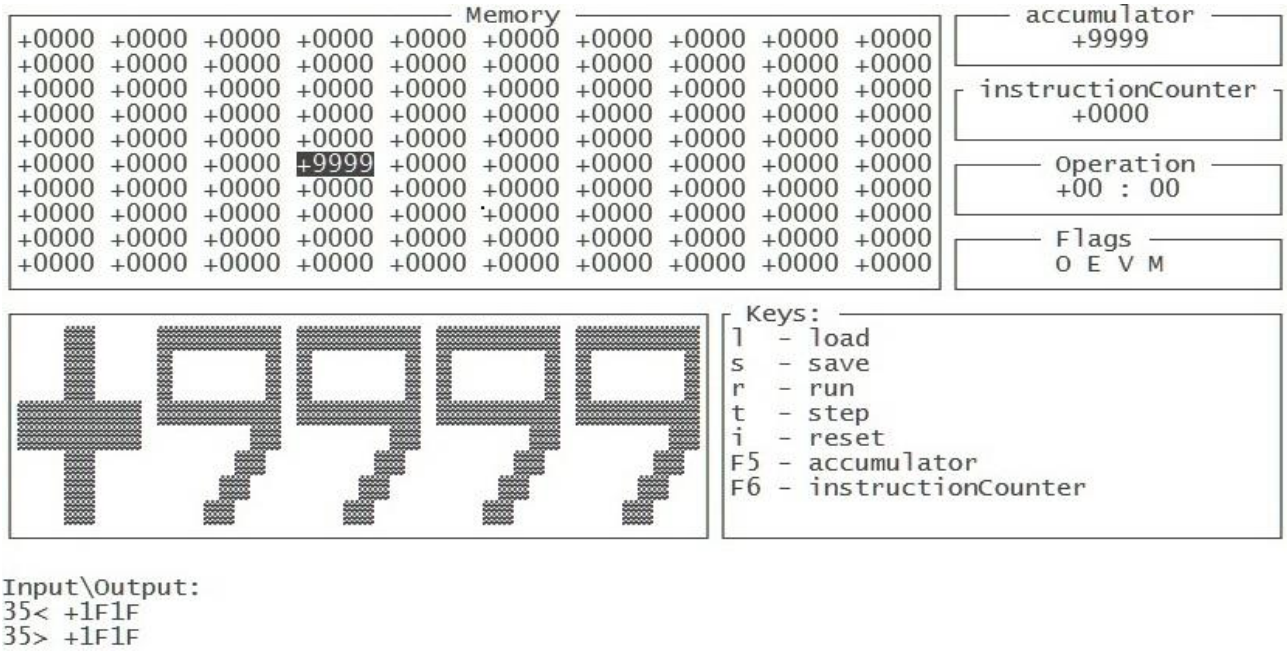



Рисунок П2.1. Интерфейс консоли управления моделью Simple Computer

### Архитектура вычислительной машины Simple Computer

Архитектура Simple Computer представлена на рисунке П2.2 и включает следующие функциональные блоки:

- оперативную память;
- внешние устройства;
- центральный процессор.

#### *Оперативная память*

Оперативная память – это часть Simple Computer, где хранятся программа и данные. Память состоит из ячеек (массив), каждая из которых хранит 15 двоичных разрядов. Ячейка – минимальная единица, к которой можно обращаться при доступе к памяти. Все ячейки последовательно пронумерованы целыми числами. Номер ячейки является её адресом и задаётся 7-разрядным числом. Предполагаем, что Simple Computer оборудован памятью из 100 ячеек (с адресами от 0 до $99_{10}$).

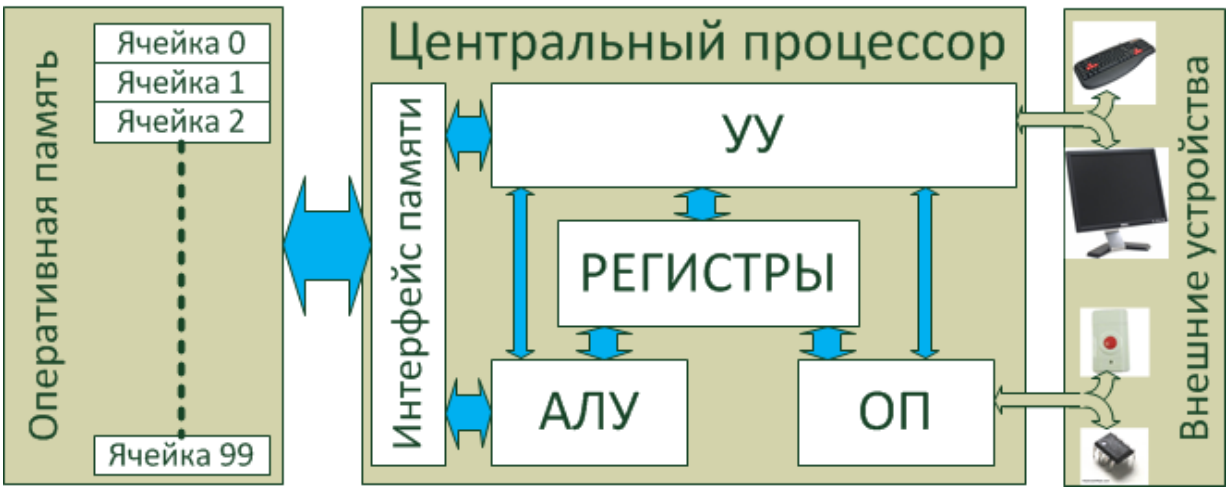



Рисунок П2.2. Архитектура вычислительной машины Simple Computer

### *Внешние устройства*

Внешние устройства включают: клавиатуру и монитор, используемые для взаимодействия с пользователем, системный таймер, задающий такты работы Simple Computer и кнопку «Reset», позволяющую сбросить Simple Computer в исходное состояние.

### *Центральный процессор*

Выполнение программ осуществляется центральным процессором Simple Computer. Процессор состоит из следующих функциональных блоков:

- регистры (аккумулятор, счётчик команд, регистр флагов);
- арифметико-логическое устройство (АЛУ);
- управляющее устройство (УУ);
- обработчик прерываний от внешних устройств (ОП);
- интерфейс доступа к оперативной памяти.

Регистры являются внутренней памятью процессора. Центральный процессор Simple Computer имеет: аккумулятор, используемый для временного хранения данных и результатов операций, счётчик команд, указывающий на адрес ячейки памяти, в которой хранится текущая выполняемая команда, и регистр флагов, сигнализирующий об определённых событиях. Аккумулятор имеет разрядность 15 бит, счетчика команд – 7 бит. Регистр флагов содержит 5 разрядов: переполнение при выполнении операции, ошибка деления на 0, ошибка выхода за границы памяти, игнорирование тактовых импульсов, указана неверная команда.

Арифметико-логическое устройство (англ. arithmetic and logic unit, ALU) – блок процессора, который служит для выполнения логических и арифметических преобразований над данными. В качестве данных могут использоваться значения, находящиеся в аккумуляторе, команды, заданные в операнде или хранящиеся в оперативной памяти. Результат выполнения операции сохраняется в аккумуляторе или может помещаться в оперативную память. В ходе выполнения операций АЛУ устанавливает значения флагов «ошибка деления на 0» и «переполнение при выполнении операции».

Управляющее устройство (англ. control unit, CU) координирует работу центрального процессора. По сути, именно это устройство отвечает за выполнение программы, записанной в оперативной памяти. В его функции входит: чтение текущей команды из памяти, её декодирование, передача номера команды и операнда в АЛУ, определение следующей выполняемой команды и реализация взаимодействий с клавиатурой и монитором. Выбор очередной команды из оперативной памяти производится по сигналу от системного таймера. Если установлен флаг «игнорирование тактовых импульсов», то эти сигналы устройством управления игнорируются. В ходе выполнения операций устройство управления устанавливает значения флагов «указана неверная команда» и «игнорирование тактовых импульсов».

Обработчик прерываний реагирует на сигналы от системного таймера и кнопки «Reset». При поступлении сигнала от кнопки «Reset» состояние процессора сбрасывается в начальное (значения всех регистров обнуляются, и устанавливается флаг «игнорирование сигналов от таймера»). При поступлении сигнала от системного таймера устройство управления начинает работать.

### *Система команд Simple Computer*

Получив текущую команду из оперативной памяти, устройство управления декодирует её с целью определить номер функции, которую надо выполнить, и операнд. Формат команды следующий (рис. П2.3): старший разряд содержит признак команды (0 – команда), разряды с 8 по 14 определяют код операции, младшие 7 разрядов содержат операнд. Коды операций, их назначение и обозначение в Simple Assembler приведены в таблице П2.1.


Рисунок П2.3. Формат команды центрального процессора Simple Computer

Таблица П2.1. Команды центрального процессора Simple Computer

| **Операция** | | **Значение** |
|---|---|---|
| **Обозначение** | **Код** | |
| **Операции ввода/вывода** | | |
| READ | 10 | Ввод с терминала в указанную ячейку памяти с контролем переполнения |
| WRITE | 11 | Вывод на терминал значения указанной ячейки памяти |
| **Операции загрузки/выгрузки в аккумулятор** | | |
| LOAD | 20 | Загрузка в аккумулятор значения из указанного адреса памяти |
| STORE | 21 | Выгрузка значения из аккумулятора по указанному адресу памяти |
| **Арифметические операции** | | |
| ADD | 30 | Выполняет сложение слова в аккумуляторе и слова из указанной ячейки памяти (результат в аккумуляторе) |

Таблица П2.1. Команды центрального процессора Simple Computer (продолжение)

| | | |
|---|---|---|
| SUB | 31 | Вычитает из слова в аккумуляторе слово из указанной ячейки памяти (результат в аккумуляторе) |
| DIVIDE | 32 | Выполняет деление слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе) |
| MUL | 33 | Вычисляет произведение слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе) |
| **Операции передачи управления** | | |
| JUMP | 40 | Переход к указанному адресу памяти |
| JNEG | 41 | Переход к указанному адресу памяти, если в аккумуляторе находится отрицательное число |
| JZ | 42 | Переход к указанному адресу памяти, если в аккумуляторе находится нуль |
| HALT | 43 | Останов, выполняется при завершении работы программы |
| **Пользовательские функции** | | |
| NOT | 51 | Двоичная инверсия слова в аккумуляторе и занесение результата в указанную ячейку памяти |
| AND | 52 | Логическая операция И между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе) |
| OR | 53 | Логическая операция ИЛИ между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе) |
| XOR | 54 | Логическая операция исключающее ИЛИ между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе) |
| JNS | 55 | Переход к указанному адресу памяти, если в аккумуляторе находится положительное число |
| JC | 56 | Переход к указанному адресу памяти, если при сложении произошло переполнение |
| JNC | 57 | Переход к указанному адресу памяти, если при сложении не произошло переполнение |
| JP | 58 | Переход к указанному адресу памяти, если результат предыдущей операции чётный |
| JNP | 59 | Переход к указанному адресу памяти, если результат предыдущей операции нечётный |
| CHL | 60 | Логический двоичный сдвиг содержимого указанной ячейки памяти влево (результат в аккумуляторе) |
| SHR | 61 | Логический двоичный сдвиг содержимого указанной ячейки памяти вправо (результат в аккумуляторе) |
| RCL | 62 | Циклический двоичный сдвиг содержимого указанной ячейки памяти влево (результат в аккумуляторе) |
| RCR | 63 | Циклический двоичный сдвиг содержимого указанной ячейки памяти вправо (результат в аккумуляторе) |
| NEG | 64 | Получение дополнительного кода содержимого указанной ячейки памяти (результат в аккумуляторе) |
| ADDC | 65 | Сложение содержимого указанной ячейки памяти с ячейкой памяти, адрес которой находится в аккумуляторе (результат в аккумуляторе) |
| SUBC | 66 | Вычитание из содержимого указанной ячейки памяти содержимого ячейки памяти, адрес которой находится в аккумуляторе (результат в аккумуляторе) |
| LOGLC | 67 | Логический двоичный сдвиг содержимого указанного участка памяти влево на количество разрядов, указанное в аккумуляторе (результат в аккумуляторе) |
| LOGRC | 68 | Логический двоичный сдвиг содержимого указанного участка памяти вправо на количество разрядов, указанное в аккумуляторе (результат в аккумуляторе) |
| RCCL | 69 | Циклический двоичный сдвиг содержимого указанного участка памяти влево на количество разрядов, указанное в аккумуляторе (результат в аккумуляторе) |
| RCCR | 70 | Циклический двоичный сдвиг содержимого указанного участка памяти вправо на количество разрядов, указанное в аккумуляторе (результат в аккумуляторе) |
| MOVA | 71 | Перемещение содержимого указанной ячейки памяти в ячейку, адрес которой указан в аккумуляторе |
| MOVR | 72 | Перемещение содержимого ячейки памяти, адрес которой содержится в аккумуляторе, в указанную ячейку памяти |
| MOVCA | 73 | Перемещение содержимого указанной ячейки памяти в ячейку памяти, адрес которой находится в ячейке памяти, на которую указывает значение аккумулятора |
| MOVCR | 74 | Перемещение в указанный участок памяти содержимого участка памяти, адрес которого находится в участке памяти, указанном в аккумуляторе |
| ADDC | 75 | Сложение содержимого указанной ячейки памяти с ячейкой памяти, адрес которой находится в ячейке памяти, указанной в аккумуляторе (результат в аккумуляторе) |
| SUBC | 76 | Вычитание из содержимого указанной ячейки памяти содержимого ячейки памяти, адрес которой находится в ячейке памяти, указанной в аккумуляторе (результат в аккумуляторе) |

### *Выполнение команд центральным процессором Simple Computer*

Команды выполняются последовательно. Адрес ячейки памяти, в которой находится текущая выполняемая команда, задаётся в регистре «Счётчик команд». Устройство управления запрашивает содержимое указанной ячейки памяти и декодирует его согласно используемому формату команд. Получив код операции, устройство управления определяет, является ли эта операция арифметико-логической. Если да, то выполнение операции передаётся в АЛУ. В противном случае операция выполняется устройством управления. Процедура выполняется до тех пор, пока флаг «игнорирование тактовых импульсов» не будет равен 1.

### *Консоль управления*

Интерфейс консоли управления представлен на рисунке П2.1. Он содержит следующие области:

- «Memory» – содержимое оперативной памяти Simple Computer;
- «Accumulator» – значение, находящееся в регистре-аккумуляторе;
- «instructionCounter» – значение регистра «счётчик команд»;
- «Operation» – результат декодирования операции;
- «Flags» – состояние регистра флагов («П» – переполнение при выполнении операции, «0» – ошибка деления на 0, «М» – ошибка выхода за границы памяти, «Т» – игнорирование тактовых импульсов, «Е» – указана неверная команда);
- «Cell» – значение выделенной ячейки памяти в области «Memory» (используется для редактирования);
- «Keys» – подсказка по функциональным клавишам;
- «Input/Output» – область, используемая Simple Computer в процессе выполнения программы для ввода информации с клавиатуры и вывода её на экран.

Содержимое ячеек памяти и регистров центрального процессора выводится в декодированном виде. При этом знак «+» соответствует значению 0 в поле «признак команды», следующие две цифры – номер команды и операнд в шестнадцатеричной системе счисления.

Пользователь имеет возможность с помощью клавиш управления курсором выбирать ячейки оперативной памяти и задавать им значения. Нажав клавишу «F5», пользователь может задать значение аккумулятора, «F6» – регистра «счётчик команд». Сохранить содержимое памяти (в бинарном виде) в файл или загрузить его обратно пользователь может, нажав на клавиши «l», «s» соответственно (после нажатия в поле «Input/Output» пользователю предлагается ввести имя файла). Запустить программу на выполнение (установить значение флага «игнорирование тактовых импульсов» в 0) можно с помощью клавиши «r». В процессе выполнения программы редактирование памяти и изменение значений регистров недоступно. Чтобы выполнить только текущую команду пользователь может нажать клавишу «t». Обнулить содержимое памяти и задать регистрам значения «по умолчанию» можно, нажав на клавишу «i».

## Лабораторная работа 1.

## Организация современных персональных компьютеров

*Цель работы*

Изучить архитектуру современных персональных компьютеров. Понять назначение основных функциональных блоков, интерфейсов их взаимодействия и схему работы центрального процессора и оперативной памяти.

*Задание на лабораторную работу*

1. Прочитайте главы 1, 2 и 3 пособия по курсу «ЭВМ и периферийные устройства».
2. Напишите реферат на одну из приведённых ниже тем согласно Вашему варианту задания. Вариант выбирается, исходя из двух последних цифр Вашего учётного имени по модулю количества тем.
3. Изучите устройство материнской платы персонального компьютера на примере любой современной модели.

*Темы рефератов*

1. Интерфейс конфигурации и управления питанием (ACPI).
2. Процессоры семейства Intel. История развития.
3. Система команд процессоров семейства Intel.
4. Микроархитектура Intel Net Burst.
5. Микроархитектура Intel Core.
6. Микроархитектура Intel Atom.
7. Микроархитектура Intel Nehalem.
8. Архитектура Intel x2APIC.
9. Технология Hyper Threading.
10. Архитектура IA64 (Itanium 2, VLIW, Intel EPIC).
11. Сравнение технологий IA32 и EM64T (Intel 64).
12. Аппаратная поддержка виртуализации в процессорах Intel (Intel VT).
13. Аппаратная поддержка виртуализации в процессорах AMD (AMD-V).
14. Расширение системы команд Streaming SIMD Extension (SSE, SSE2, SSE3, SSE4).
15. Расширение системы команд MMX, 3DNow.
16. Интерфейс PCI, PCI-Express.
17. Интерфейс USB.
18. Интерфейс Bluetooth.
19. Интерфейсы D-Sub и DVI.
20. Интерфейс IEEE1394 (FireWire).
21. Принципы работы процессорного кэша.
22. Предсказание переходов. Спекулятивное выполнение.
23. Параллелизм уровня команд (Instruction Level Parallelism, ILP). Конвейеризация.
24. Параллелизм уровня потоков (Thread Level Parallelism). Технология.
25. Параллелизм уровня заданий. Многоядерные процессоры.

26. Шина Hyper Transport.
27. Внешняя память. Дисковые массивы RAID.
28. Оперативная память. Чип SPD.
29. Набор микросхем системной логики. Архитектура. Примеры существующих чипсетов.

*Процедура защиты реферата*

Защита реферата производится в сроки, указанные в календарном плане и проводится в два этапа:

1) беседа по представленному в реферате материалу;
2) рассказ «устройство узлов персонального компьютера».

На втором этапе студентам предоставляется один из узлов персонального компьютера и требуется рассказать назначение этого устройства и его основные характеристики.

*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.

## Разработка библиотеки mySimpleComputer. Оперативная память, регистр флагов, декодирование операций

*Цель работы*

Изучить принципы работы оперативной памяти. Познакомиться с разрядными операциями языка Си. Разработать библиотеку mySimpleComputer, включающую функции по декодированию команд, управлению регистрами и взаимодействию с оперативной памятью.

*Задание на лабораторную работу*

1. Прочитайте главу 4 пособия по курсу «ЭВМ и периферийные устройства». Изучите принципы работы разрядных операций в языке Си: как можно изменить значение указанного разряда целой переменной или получить его значение. Вспомните, как сохранять информацию в файл и считывать её оттуда в бинарном виде.
2. Разработайте функции по взаимодействию с оперативной памятью, управлению регистром флагов и кодированию/декодированию команд:

a. `int sc_memoryInit ()` инициализирует оперативную память Simple Computer, задавая всем её ячейкам нулевые значения. В качестве оперативной памяти используется массив целых чисел, определённый статически в рамках библиотеки. Размер массива равен 100 элементам;
b. `int sc_memorySet (int address, int value)` задаёт значение указанной ячейки памяти как `value`. Если адрес выходит за допустимые границы, то устанавливается флаг «ошибка выхода за границы памяти», и работа функции прекращается с ошибкой;
c. `int sc_memoryGet (int address, int *value)` возвращает значение указанной ячейки памяти в `value`. Если адрес выходит за допустимые границы, то устанавливается флаг «ошибка выхода за границы памяти», и работа функции прекращается с ошибкой. Значение `value` в этом случае не изменяется;
d. `int sc_memorySave (char *filename)` сохраняет содержимое памяти в файл в бинарном виде (используя функцию `write` или `fwrite`);
e. `int sc_memoryLoad (char *filename)` загружает из указанного файла содержимое оперативной памяти (используя функцию `read` или `fread`);
f. `int sc_regInit (void)` инициализирует регистр флагов нулевым значением;
g. `int sc_regSet (int register, int value)` устанавливает значение указанного регистра флагов. Для номеров регистров флагов должны использоваться маски, задаваемые макросами (`#define`). Если указан недопустимый номер регистра или некорректное значение, то функция завершается с ошибкой;
h. `int sc_regGet (int register, int *value)` возвращает значение указанного флага. Если указан недопустимый номер регистра, то функция завершается с ошибкой;
i. `int sc_commandEncode (int command, int operand, int *value)` кодирует команду с указанным номером и операндом и помещает результат в `value`. Если указаны неправильные значения для команды или операнда, то функция завершается с ошибкой. В этом случае значение `value` не изменяется;
j. `int sc_commandDecode (int value, int *command, int *operand)` декодирует значение как команду Simple Computer. Если декодирование невозможно, то устанавливается флаг E «указана неверная команда», и функция завершается с ошибкой.

3. Оформите разработанные функции как статическую библиотеку. Подготовьте заголовочный файл для неё.

*Защита лабораторной работы*

Для защиты лабораторной работы необходимо подготовить программу, демонстрирующую использование созданной библиотеки функций (сборка программы с библиотекой, использование заголовочного файла, примеры вызовов каждой функции, проверка корректности работы функций при различных входных значениях).

*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.

## Консоль управления моделью Simple Computer. Текстовая часть

*Цель работы*

Изучить принципы работы терминалов ЭВМ в текстовом режиме. Понять, каким образом кодируется текстовая информация и как с её помощью можно управлять работой терминалов. Разработать библиотеку функций myTerm, включающую базовые функции по управлению текстовым терминалом (очистка экрана, позиционирование курсора, управления цветом). Начать разрабатывать консоль управления Simple Computer (вывести на экран текстовую часть).

*Задание на лабораторную работу*

1. Прочитайте главу 5 пособия по курсу «ЭВМ и периферийные устройства». Обратите особое внимание на параграфы 5.4 и 5.5. Изучите страницу `man` для команды `infocmp`, базы `terminfo`, функции `ioctl`.
2. Откройте текстовый терминал и запустите оболочку `bash` (оболочка запускается автоматически). Используя команду `infocmp`, определите (и перепишите их себе) escape-последовательности для терминала, выполняющие следующие действия:
    - очистка экрана и перемещение курсора в левый верхний угол (clear_screen);
    - перемещение курсора в заданную позицию экрана (cursor_address);
    - задание цвета последующих выводимых символов (set_a_background);
    - определение цвета фона для последующих выводимых символов (set_a_foreground);
    - скрытие и восстановление курсора (cursor_invisible, cursor_visible).
3. Используя оболочку `bash`, команду `echo -e` и скрипт[2], проверьте работу полученных последовательностей. Символ «Escape» задаётся как \033 или \E. Например: `echo -e "\033[m"`. Для проверки сформируйте последовательность escape-команд, выполняющую следующие действия:
    - очищает экран;
    - выводит в пятой строке, начиная с 10-го символа Ваше имя красными буквами на чёрном фоне;
    - выводит в шестой строке, начиная с 8-го символа Вашу группу зелёным цветом на белом фоне;
    - перемещает курсор в 10-ую строку, 1-й символ и возвращает настройки цвета в значения «по умолчанию».
4. Разработать следующие функции:
    - `int mt_clrscr (void)` производит очистку и перемещение курсора в левый верхний угол экрана;
    - `int mt_gotoXY (int, int)` перемещает курсор в указанную позицию. Первый параметр – номер строки, второй – номер столбца;
    - `int mt_getscreensize (int *rows, int *cols)` определяет размер экрана терминала (количество строк и столбцов);
    - `int mt_setfgcolor (enum colors)` устанавливает цвет последующих выводимых символов. В качестве параметра передаётся константа из созданного Вами перечислимого типа `colors`, описывающего цвета терминала;
    - `int mt_setbgcolor (enum colors)` устанавливает цвет фона последующих выводимых символов. В качестве параметра пере-

[2] Скрипт – это текстовый файл, содержащий команды оболочки. Запускается на выполнение командой bash имя_файла.

даётся константа из созданного Вами перечислимого типа `colors`, описывающего цвета терминала.

Все функции возвращают 0 в случае успешного выполнения и –1 в случае ошибки. В качестве терминала используется стандартный поток вывода.

5. Оформите разработанные функции как статическую библиотеку myTerm. Подготовьте заголовочный файл для неё.

*Защита лабораторной работы*

Для защиты лабораторной работы необходимо подготовить программу, демонстрирующую использование созданной библиотеки функций (сборка программы с библиотекой, использование заголовочного файла, примеры вызовов каждой функции, проверка корректности работы функций при различных входных значениях), а также программу, выводящую на экран согласно рисунку П2.1 содержимое оперативной памяти, регистров и назначение клавиш.

*Контрольные вопросы*

1. Взаимодействие с устройствами в Linux. Специальные файлы устройств.
2. Функции open, close, read, write.
3. Терминалы. Типы терминалов. Эмуляция терминала. Режимы работы.
4. Управление терминалом. Команды. Низкоуровневое управление.
5. Что такое escape-последовательность?
6. Как определить escape-последовательности для терминала?

## Лабораторная работа 4.

## Консоль управления моделью Simple Computer. Псевдографика. «Большие символы»

*Цель работы*

Изучить работу текстового терминала с псевдографическими символами. Понять, что такое шрифт и как он используется в терминалах при выводе информации. Разработать библиотеку myBigChars, реализующую функции по работе с псевдографикой и выводу «больших символов» на экран. Доработать консоль управления Simple Computer так, чтобы выводились псевдографические элементы.

*Задание на лабораторную работу*

1. Прочитайте главу 5 пособия по курсу «ЭВМ и периферийные устройства». Обратите особое внимание на параграфы 5.2, 5.3, 5.4.2. Изучите страницу `man` для команды `infocmp`, базы `terminfo` (раздел псевдографика).
2. Используя оболочку `bash` и команду `infocmp`, определите escape-последовательности для переключения используемых терминалом коди-

ровочных таблиц (enter_alt_charset_mode и exit_alt_charset_mode) и соответствие символов для вывода псевдографики (acs_chars).

3. Используя оболочку `bash`, команду `echo -e` и скрипт, проверьте работу полученных последовательностей. Символ «Escape» задаётся как \033 или \E. Например: `echo -e "\033[m"`. Для проверки сформируйте последовательность escape-команд, выполняющую следующие действия:
   - очищает экран;
   - выводит псевдографическую рамку, начиная с 5-го символа 10-ой строки, размером 8 строк на 8 столбцов;
   - с помощью псевдографического символа «закрашенный прямоугольник» (ACS_CKBOARD) в рамке выводится «большой символ», соответствующий последней цифре дня Вашего рождения (например, день рождения 13 января 1991 года, выводится цифра 3).
4. Разработать следующие функции:
   - `int bc_printA (char *str)` выводит строку символов с использованием дополнительной кодировочной таблицы;
   - `int bc_box(int x1, int y1, int x2, int y2)` выводит на экран псевдографическую рамку, в которой левый верхний угол располагается в строке `x1` и столбце `y1`, а её ширина и высота равны `y2` столбцов и `x2` строк;
   - `int bc_printbigchar (int [2], int x, int y, enum colors, enum colors)` выводит на экран «большой символ» размером восемь строк на восемь столбцов, левый верхний угол которого располагается в строке `x` и столбце `y`. Третий и четвёртый параметры определяют цвет и фон выводимых символов. «Символ» выводится, исходя из значений массива целых чисел, следующим образом. В первой строке выводится 8 младших бит первого числа, во второй – следующие 8, в третьей и 4-ой – следующие. В 5-ой строке выводятся 8 младших бит второго числа и т.д. При этом если значение бита равно 0, то выводится символ «пробел», иначе – символ, закрашивающий знакоместо (ACS_CKBOARD);
   - `int bc_setbigcharpos (int *big, int x, int y, int value)` устанавливает значение знакоместа «большого символа» в строке `x` и столбце `y` в значение `value`;
   - `int bc_getbigcharpos(int *big, int x, int y, int *value)` возвращает значение позиции в «большом символе» в строке `x` и столбце `y`;
   - `int bc_bigcharwrite (int fd, int *big,int count)` записывает заданное число «больших символов» в файл. Формат записи определяется пользователем;
   - `int bc_bigcharread (int fd, int *big, int need_count, int *count)` считывает из файла заданное

количество «больших символов». Третий параметр указывает адрес переменной, в которую помещается количество считанных символов или 0 в случае ошибки.

Все функции возвращают 0 в случае успешного выполнения и –1 в случае ошибки. В качестве терминала используется стандартный поток вывода.

5. Оформите разработанные функции как статическую библиотеку myBigChars. Подготовьте заголовочный файл для неё.

*Защита лабораторной работы*

Для защиты лабораторной работы необходимо подготовить программу, демонстрирующую использование созданной библиотеки функций (сборка программы с библиотекой, использование заголовочного файла, примеры вызовов каждой функции, проверка корректности работы функций при различных входных значениях). Необходимо доработать программу лабораторной работы № 3, выводящую на экран согласно рисунку П2.1 содержимое оперативной памяти, регистров и назначение клавиш, так, чтобы на экране были нарисованы рамки, и выводилось «большими символами» содержимое ячейки памяти, на которую указывает регистр «instructionCounter».

*Контрольные вопросы*

7. Что такое шрифт? Как он используется при выводе символов на экран?
8. Зачем используется кодировочная таблица символов? Какие таблицы Вы знаете?
9. Почему символы, рисующие рамку в текстовом режиме, называются «псевдографическими»?

## Лабораторная работа 5.

## Консоль управления моделью Simple Computer. Клавиатура. Обработка нажатия клавиш. Неканонический режим работы терминала

*Цель работы*

Изучить устройство клавиатуры и принципы обработки нажатия клавиш в текстовом терминале. Создать «распознаватель» нажатой клавиши по формируемой последовательности символов. Разработать библиотеку myReadkey. Доработать интерфейс консоли управления Simple Computer так, чтобы можно было изменять значения ячеек памяти и регистров.

*Задание на лабораторную работу*

1. Прочитайте главу 5 пособия по курсу «ЭВМ и периферийные устройства». Обратите особое внимание на параграф 5.1. Изучите страницу `man` для команд `infocmp` и `read`, базы `terminfo`.
2. Используя оболочку `bash` и команду `read`, определите последовательности, формируемые нажатием на буквенно-цифровые, функциональные

клавиши и клавиши управления курсором. Используя команду `infocmp`, убедитесь, что получены правильные последовательности символов, генерируемые функциональными клавишами «F5» и «F6».

3. Разработайте функции:
   - `int rk_readkey (enum keys *)` анализирует последовательность символов (возвращаемых функцией `read` при чтении с терминала) и возвращает первую клавишу, которую нажал пользователь. В качестве параметра в функцию передаётся адрес переменной, в которую возвращается номер нажатой клавиши (`enum keys` – перечисление распознаваемых клавиш);
   - `int rk_mytermsave (void)` сохраняет текущие параметры терминала;
   - `int rk_mytermrestore (void)` восстанавливает сохранённые параметры терминала;
   - `int rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint)` переключает терминал между режимами. Для неканонического режима используются значения второго и последующего параметров.
4. Оформите разработанные функции как статическую библиотеку myReadkey. Подготовьте заголовочный файл для неё.

*Защита лабораторной работы*

Для защиты лабораторной работы необходимо подготовить программу, демонстрирующую использование созданной библиотеки функций (сборка программы с библиотекой, использование заголовочного файла, примеры вызовов каждой функции, проверка корректности работы функций при различных входных значениях). Необходимо доработать программу лабораторной работы № 3, выводящую на экран согласно рисунку П2.1 консоль управления Simple Computer так, чтобы можно было задавать значения ячейкам оперативной памяти, регистрам, и обрабатывалось нажатие клавиш «s», «l».

*Контрольные вопросы*

1. Режимы работы терминала. Как настроить терминал для работы в неканоническом режиме?
2. Работа с терминалом в Linux. Структура termios.

## Лабораторная работа 6.

## Подсистема прерываний ЭВМ. Сигналы и их обработка

*Цель работы*

Изучить принципы работы подсистемы прерываний ЭВМ. Понять, как обрабатываются сигналы в Linux. Реализовать обработчик прерываний в модели Simple Computer. Доработать модель Simple Computer, создав обработчик

прерываний от внешних устройств «системный таймер» и «кнопка».

*Задание на лабораторную работу*

1. Прочитайте главу 6 пособия по курсу «ЭВМ и периферийные устройства». Изучите страницу `man` для функций `signal`, `setitimer`.
2. Доработайте консоль Simple Computer. Создайте обработчик прерываний от системного таймера так, чтобы при каждом его срабатывании при нулевом значении флага «игнорирование тактовых импульсов» значение регистра «instructionCounter» увеличивалось на 1, а при поступлении сигнала SIGUSR1 состояние Simple Computer возвращалось в исходное. Обработка нажатых клавиш осуществляется только в случае, если сигналы от таймера не игнорируются.

*Защита лабораторной работы*

Для защиты лабораторной работы необходимо подготовить программу, реализующую консоль управления Simple Computer и демонстрирующую работу обработчика прерываний.

*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 7.

## Устройство хранения данных на жестких магнитных дисках

*Цель работы*

Изучить устройство накопителей на жёстких магнитных дисках. Разработать библиотеку функций по преобразованию геометрий и адресов секторов накопителей на жёстких магнитных дисках. Создать программу, рассчитывающую таблицу разделов (MBR).

*Задание на лабораторную работу*

1. Прочитайте главу 7 пособия по курсу «ЭВМ и периферийные устройства».
2. Разработайте пользовательские типы (`typedef`) для хранения адресов секторов и геометрий жёстких дисков в форматах:
   - CHS (20 бит). Имя пользовательского типа – `tCHS`;
   - ECHS или Large (24 бита). Имя пользовательского типа – `tLARGE`;

- CHS из стандарта IDE (28 бит). Имя пользовательского типа `tIDECHS`;
- LBA (32 бит). Имя пользовательского типа – `tLBA`.

3. Создайте библиотеку функций по преобразованию геометрий и адресов секторов накопителей на жёстких магнитных дисках в разные стандарты:
   - `int g_lba2chs (tLBA, tCHS *)`
   - `int g_lba2large (tLBA, tLARGE *)`
   - `int g_lba2idechs (tLBA, tIDECHS *)`
   - `int g_chs2large (tCHS, tLARGE *)`
   - `int g_chs2lba (tCHS, tLBA *)`
   - `int g_chs2idechs (tIDECHS, tLBA *)`
   - `int g_large2chs (tLARGE, tCHS *)`
   - `int g_large2idechs (tLARGE, tIDECHS *)`
   - `int g_large2lba (tLARGE, tLBA *)`
   - `int g_idechs2chs (tIDECHS. tCHS *)`
   - `int g_idechs2lagre (tIDECHS. tLARGE *)`
   - `int g_idechs2lba (tIDECHS. tLBA *)`
   - `int a_lba2chs (tCHS geometry, tLBA, tCHS *)`
   - `int a_lba2large (tLARGE geometry, tLBA, tLARGE *)`
   - `int a_lba2idechs (tIDECHS geometry, tLBA, tIDECHS *).`
   - `int a_chs2lba (tCHS geometry, tCHS, tLBA *)`
   - `int a_large2lba (tLARGE geometry, tLARGE, tLBA *)`
   - `int a_idechs2lba (tIDECHS geometry, tIDECHS, tLBA *)`
   - `int a_large2chs (tLARGE geometry1, tCHS geometry2, tLARGE, tCHS *)`
   - `int a_large2idechs (tLARGE geometry1, tIDECHS geometry2, tLARGE, tIDECHS *)`
   - `int a_chs2large (tCHS geometry1, tLARGE geometry2, tCHS, tLARGE *)`
   - `int a_idechs2large (tIDECHS geometry1, tLARGE geometry2, tIDECHS, tLARGE *)`
   - `int a_chs2idechs (tCHS geometry1, tIDECHS geometry2, tCHS, tIDECHS *)`

```
• int a_idechs2chs (tIDECHS geometry1,
                    tCHS geometry2, tIDECHS,
                    tCHS *)
```

*Защита лабораторной работы*

С использованием библиотеки функций необходимо разработать программу, выполняющую следующие действия:

- предлагает пользователю ввести геометрию диска в формате IDECHS;
- рассчитывает размер жёсткого диска в ГБ и выводит его на экран;
- предлагает пользователю ввести: размер требуемого раздела на диске, его тип, и будет ли он активным (активным может быть только один раздел на диске!);
- на основании введённых данных рассчитывает строку в таблице разделов. Считается, что первый создаваемый пользователем раздел располагается, начиная с сектора 1 (LBA), второй – следом за ним, третий – следом за вторым и т.д.;
- формирование таблицы разделов прекращается, если пользователь ввёл 0 (нуль) как размер раздела, или на диске больше не осталось свободного места;
- после ввода всей требуемой информации формируются таблицы разделов (основная и все расширенные) и выводятся на экран с указанием номера сектора, в котором будет записана каждая таблица.

*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.

# ПРИЛОЖЕНИЕ 3. ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

В рамках курсовой работы необходимо доработать модель Simple Computer так, чтобы она обрабатывала команды, записанные в оперативной памяти. Система команд представлена в таблице П2.1. Из пользовательских функций необходимо реализовать только одну согласно варианту задания (номеру Вашей учётной записи). Для разработки программ требуется создать трансляторы с языков Simple Assembler и Simple Basic.

### Обработка команд центральным процессором

Для выполнения программ моделью Simple Computer необходимо реализовать две функции:

- `int ALU (int command, int operand)` реализует алгоритм работы арифметико-логического устройства. Если при выполнении функции возникла ошибка, которая не позволяет дальше выполнять программу, то функция возвращает –1, иначе – 0;
- `int CU (void)` обеспечивает работу устройства управления.

Обработку команд осуществляет устройство управления. Функция CU вызывается либо обработчиком сигнала от системного таймера, если не установлен флаг «игнорирование тактовых импульсов», либо при нажатии на клавишу «t». Алгоритм работы функции следующий:

1. Из оперативной памяти считывается ячейка, адрес которой хранится в регистре «instructionCounter».
2. Полученное значение декодируется как команда.
3. Если декодирование невозможно, то устанавливаются флаги «указана неверная команда» и «игнорирование тактовых импульсов» (системный таймер можно отключить), и работа функции прекращается.
4. Если получена арифметическая или логическая операция, то вызывается функция ALU, иначе команда выполняется самим устройством управления.
5. Определяется, какая команда должна быть выполнена следующей, и адрес её ячейки памяти заносится в регистр «instructionCounter».
6. Работа функции завершается.

### Транслятор с языка Simple Assembler

Разработка программ для Simple Computer может осуществляться с использованием низкоуровневого языка Simple Assembler. Для того чтобы программа могла быть обработана Simple Computer, необходимо реализовать транслятор, переводящий текст Simple Assembler в бинарный формат, который может быть считан консолью управления. Пример программы на Simple Assembler:

```
00  READ   09      ; (Ввод A)
01  READ   10      ; (Ввод B)
02  LOAD   09      ; (Загрузка A в аккумулятор)
03  SUB    10      ; (Отнять B)
04  JNEG   07      ; (Переход на 07, если отрицательное)
```

```
05  WRITE 09      ; (Вывод A)
06  HALT  00      ; (Останов)
07  WRITE 10      ; (Вывод B)
08  HALT  00      ; (Останов)
09  =     +0000   ; (Переменная A)
10  =     +9999   ; (Переменная B)
```

Программа транслируется по строкам, задающим значение одной ячейки памяти. Каждая строка состоит как минимум из трёх полей: адрес ячейки памяти, команда (символьное обозначение), операнд. Четвертым полем может быть указан комментарий, который обязательно должен начинаться с символа «точка с запятой». Название команд представлено в таблице П2.1. Дополнительно используется команда «=», которая явно задаёт значение ячейки памяти в формате вывода его на экран консоли (+XXXX).

Команда запуска транслятора должна иметь вид: `sat файл.sa файл.o`, где `файл.sa` – имя файла, в котором содержится программа на Simple Assembler, `файл.o` – результат трансляции.

### Транслятор с языка Simple Basic

Для упрощения программирования пользователю модели Simple Computer должен быть предоставлен транслятор с высокоуровневого языка Simple Basic. Файл, содержащий программу на Simple Basic, преобразуется в файл с кодом Simple Assembler. Затем файл на Simple Assembler транслируется в бинарный формат.

В языке Simple Basic используются следующие операторы: REM, INPUT, OUTPUT, GOTO, IF, LET, END. Пример программы на Simple Basic:

```
10  REM Это комментарий
20  INPUT A
30  INPUT B
40  LET C = A – B
50  IF C < 0 GOTO 20
60  PRINT C
70  END
```

Каждая строка программы состоит из номера строки, оператора Simple Basic и параметров. Номера строк должны следовать в возрастающем порядке. Все команды за исключением команды конца программы могут встречаться в программе многократно. Simple Basic должен оперировать с целыми выражениями, включающими операции +, –, *, и /. Приоритет операций аналогичен приоритету в языке Си. Для того чтобы изменить порядок вычисления, можно использовать скобки.

Транслятор должен распознавать только буквы верхнего регистра, то есть все символы в программе на Simple Basic должны быть набраны в верхнем регистре (символ нижнего регистра приведёт к ошибке). Имя переменной может состоять только из одной буквы. Simple Basic оперирует только с целыми значениями переменных, в нём отсутствует объявление переменных, а упоминание

переменной автоматически вызывает её объявление и присваивает ей нулевое значение. Синтаксис языка не позволяет выполнять операции со строками.

## Оформление отчёта по курсовой работе

Отчёт по курсовой работе представляется в виде пояснительной записки (ПЗ), к которой прилагается диск с разработанным программным обеспечением. В пояснительную записку должны входить:

- титульный лист;
- полный текст задания по курсовой работе;
- реферат (объём ПЗ, количество таблиц, рисунков, схем, программ, приложений, краткая характеристика и результаты работы);
- содержание:
  - · постановка задачи исследования;
  - · блок-схемы используемых алгоритмов;
  - · программная реализация;
  - · результаты проведённого исследования;
  - · выводы;
- список использованной литературы;
- подпись, дата.

Пояснительная записка должна быть оформлена на листах формата А4, имеющих поля. Все листы следует сброшюровать и пронумеровать.

Сергей Николаевич Мамойленко
Ольга Владимировна Молдованова

# ЭВМ и периферийные устройства

Учебное пособие

Редактор: Ю.С.Майданов
Корректор: В.В.Сиделина

---

Подписано в печать 12.04.2012г.
Формат бумаги 60 х 84/16, отпечатано на ризографе, шрифт № 10,
изд. л.6,6, заказ № 30, тираж – 100 экз., СибГУТИ.
630102, г. Новосибирск, ул. Кирова, д. 86

106