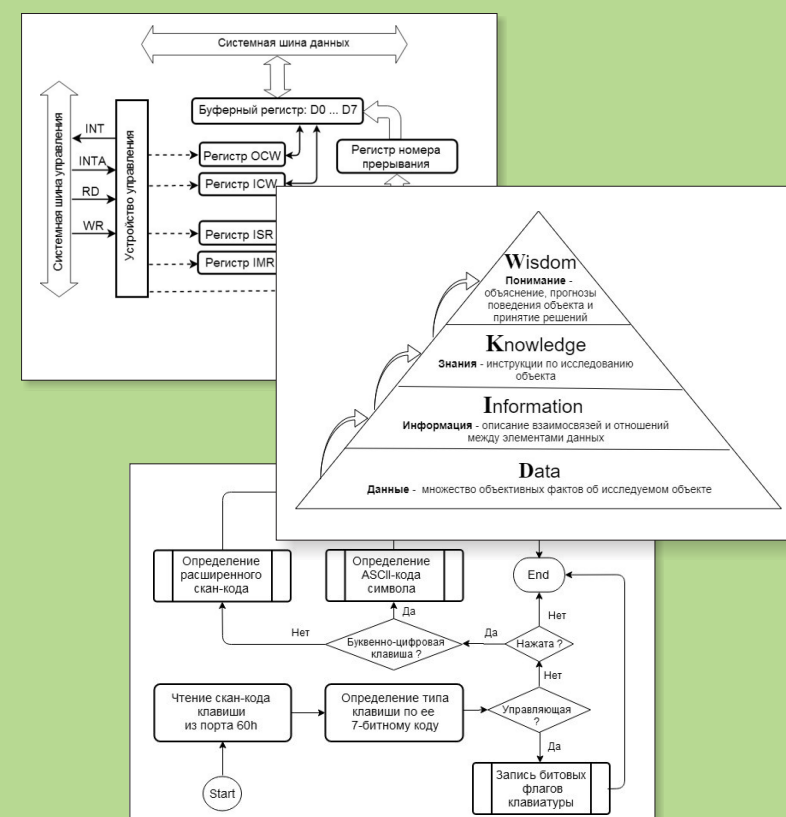


В. К. Волк

ИНФОРМАТИКА

ВВОДНЫЙ КУРС ДЛЯ СТУДЕНТОВ ИТ-СПЕЦИАЛЬНОСТЕЙ



ИНФОРМАТИКА

В.К.Волк

УЧЕБНОЕ ПОСОБИЕ

ISBN 978-5-4217-0548-2



Курганский
государственный
университет



Библиотечно-издательский
центр
65-48-12

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курганский государственный университет»

В. К. Волк

ИНФОРМАТИКА

**ВВОДНЫЙ КУРС
ДЛЯ СТУДЕНТОВ ИТ-СПЕЦИАЛЬНОСТЕЙ**

Учебное пособие

Курган 2020

УДК 681.3(075.8)

ББК 32.81я73

В 67

Рецензенты:

канд. физ.-мат. наук, профессор кафедры прикладной информатики и автоматизации бизнес-процессов Шадринского государственного педагогического университета В. Ю. Пирогов;

канд. с.-хоз. наук кафедры физики, математики и информационных технологий Курганской государственной сельскохозяйственной академии им. Т. С. Мальцева А. А. Битюгина.

Печатается по решению методического совета Курганского государственного университета.

Научный редактор – канд. пед. наук, доцент А. А. Медведев.

Волк, В. К.

Информатика. Вводный курс для студентов IT-специальностей : учебное пособие / В. К. Волк. – Курган : Изд-во Курганского гос. ун-та, 2020. – 218 с. – URL: <http://dSPACE.KGSU.RU/xmlui>. – Текст : электронный.

Учебное пособие состоит из восьми глав, в которых дается введение в прикладные аспекты информатики и рассматриваются алгоритмы взаимодействия программных и аппаратных компонентов простейшей ЭВМ.

В первых трех главах обсуждается содержание информатики как научно-технической дисциплины, ее взаимосвязи с другими дисциплинами и информационными технологиями, рассматриваются определения понятия «информация», ее свойства и единицы измерения, а также методы компьютерного представления информации: системы счисления, основы двоичной и шестнадцатеричной арифметики, двоичное кодирование текстовой, числовой и (частично) графической информации.

В четвертой и пятой главах рассмотрен программно-аппаратный комплекс простейшей компьютерной системы: базовые принципы Фон-Неймана, состав компонентов, структура адресного пространства и схема взаимодействия центрального процессора с периферийным оборудованием. В шестой и седьмой главах рассмотрены основы функционирования файловых систем и видеосистемы компьютера, а также процесс ввода данных с клавиатуры.

Восьмая глава пособия – это лабораторный практикум, содержащий задания экспериментального характера по материалу шестой и седьмой глав. Выполнение заданий нацелено на практическое освоение алгоритмов работы основных подсистем компьютера и потребует использования соответствующих инструментальных программных средств.

Каждая глава завершается перечнем контрольных вопросов и практических заданий различных уровней сложности. Пособие предназначено для студентов младших курсов IT-специальностей и может быть использовано при самостоятельной подготовке, а также при подготовке и проведении лекционных курсов, практических и лабораторных занятий.

ISBN 978-5-4217-0548-2

© Курганский
государственный университет, 2020
© Волк В. К., 2020

Содержание

ПРЕДИСЛОВИЕ.....	6
ГЛАВА 1. ИНФОРМАТИКА – ИСТОРИЯ СТАНОВЛЕНИЯ	7
ГЛАВА 2. ИНФОРМАЦИЯ.....	16
2.1 ПОНЯТИЕ ИНФОРМАЦИИ.....	16
2.2 СВОЙСТВА ИНФОРМАЦИИ.....	18
2.3 КОЛИЧЕСТВО ИНФОРМАЦИИ.....	20
2.4 ИНФОРМАЦИЯ, ДАННЫЕ И ЗНАНИЯ – МОДЕЛЬ DIKW	23
2.5 КОНТРОЛЬНЫЕ ЗАДАНИЯ	27
ГЛАВА 3. ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В КОМПЬЮТЕРНЫХ СИСТЕМАХ.....	31
3.1 ЧИСЛОВОЕ КОДИРОВАНИЕ ДАННЫХ	31
3.2 СИСТЕМЫ СЧИСЛЕНИЯ.....	32
3.2.1 Аддитивные системы счисления	32
3.2.2 Позиционные системы счисления.....	35
3.3 ДВОИЧНОЕ КОДИРОВАНИЕ ТЕКСТОВОЙ ИНФОРМАЦИИ	43
3.3.1 Стандарт ASCII	44
3.3.2 Стандарт UNICODE.....	47
3.4 ДВОИЧНОЕ КОДИРОВАНИЕ ДЕСЯТИЧНЫХ ЧИСЕЛ.....	50
3.4.1 Кодирование натуральных чисел	51
3.4.2 Кодирование целых чисел со знаком.....	52
3.4.3 Кодирование вещественных чисел.....	58
3.5 КОНТРОЛЬНЫЕ ЗАДАНИЯ	63
ГЛАВА 4. АППАРАТНОЕ ОБЕСПЕЧЕНИЕ ЭВМ.....	68
4.1 БАЗОВЫЕ ПРИНЦИПЫ ФУНКЦИОНИРОВАНИЯ ЭВМ.....	68
4.2 ТИПОВАЯ АРХИТЕКТУРА ПРОСТЕЙШЕЙ ЭВМ	71
4.3 ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР	74
4.4 ЗАПОМИНАЮЩИЕ УСТРОЙСТВА	78
4.5 ПЕРИФЕРИЙНОЕ ОБОРУДОВАНИЕ.....	86
4.6 АДРЕСНОЕ ПРОСТРАНСТВО ПК.....	86

4.6.1	Сегментная организация адресного пространства основной памяти.....	87
4.6.2	Стандартное распределение базового адресного пространства	91
4.6.3	Адресное пространство ввода-вывода.....	95
4.7	КОНТРОЛЬНЫЕ ЗАДАНИЯ	99
ГЛАВА 5. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ		101
5.1	КЛАССИФИКАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.....	101
5.2	СИСТЕМНОЕ ПО	102
5.2.1	Программная структура MS DOS.....	102
5.2.2	Процесс загрузки MS DOS.....	104
5.2.3	Функциональная структура MS DOS.....	107
5.3	КОНТРОЛЬНЫЕ ЗАДАНИЯ	111
ГЛАВА 6. ФАЙЛОВЫЕ СИСТЕМЫ		112
6.1	ТРЕХУРОВНЕВАЯ МОДЕЛЬ ДИСКОВОГО ПРОСТРАНСТВА.....	112
6.2	ФАЙЛОВЫЕ FAT-СИСТЕМЫ.....	115
6.2.1	Структура системной области тома	116
6.2.2	Алгоритмы выполнения типовых файловых операций	124
6.2.3	Недостатки FAT-систем	136
6.3	ФАЙЛОВЫЕ NTFS-СИСТЕМЫ	138
6.3.1	Структура тома NTFS.....	139
6.3.2	Мета-файлы NTFS	139
6.3.3	Схемы хранения файлов.....	141
6.3.4	Схемы хранения каталогов	144
6.3.5	Атрибуты файлов и каталогов.....	147
6.4	КОНТРОЛЬНЫЕ ЗАДАНИЯ	148
ГЛАВА 7. ОБМЕН ДАННЫМИ С ПЕРИФЕРИЙНЫМИ УСТРОЙСТВАМИ		152
7.1	СИСТЕМА ОБРАБОТКИ ПРЕРЫВАНИЙ	152
7.1.1	Аппаратные прерывания	153
7.1.2	Программные прерывания	153
7.1.3	Контроллер прерываний.....	154

7.1.4 Таблица векторов прерываний	158
7.1.5 Процедура обработки прерывания.....	159
7.2 КЛАВИАТУРА	161
7.2.1 Контроллер клавиатуры	161
7.2.2 Алгоритм обработки прерывания клавиатуры	164
7.3 ВИДЕОСИСТЕМА ПК	177
7.3.1 Аппаратный комплекс видеосистемы ПК.....	178
7.3.2 Программное обеспечение видеосистемы ПК.....	180
7.3.3 Структуры данных, используемые видеосистемой ПК	181
7.3.4 Кодирование данных в видеопамати	183
7.3.5 Знакогенераторы	187
7.4 КОНТРОЛЬНЫЕ ЗАДАНИЯ	189
ГЛАВА 8. ЛАБОРАТОРНЫЙ ПРАКТИКУМ	192
ОБЩИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ	192
ЛАБОРАТОРНАЯ РАБОТА №1. КОМАНДНЫЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ ПК	194
ЛАБОРАТОРНАЯ РАБОТА №2. ПРОГРАММИРОВАНИЕ ПАКЕТНЫХ ФАЙЛОВ	202
ЛАБОРАТОРНАЯ РАБОТА №3. ФАЙЛОВАЯ СИСТЕМА ПК	209
ЛАБОРАТОРНАЯ РАБОТА №4. ИССЛЕДОВАНИЕ АДРЕСНОГО ПРОСТРАНСТВА ПК	212
ЛАБОРАТОРНАЯ РАБОТА №5. КЛАВИАТУРА ПК	213
ЛАБОРАТОРНАЯ РАБОТА №6. ВИДЕОСИСТЕМА ПК	215
Приложение А. Инструкция по установке виртуальной DOS-машины	217

ПРЕДИСЛОВИЕ

Дисциплина «Информатика» прочно закрепилась в вузовских образовательных программах как технических, так и гуманитарных направлений, и занимает в них достойное место в одном ряду с другими общеобразовательными математическими и естественно-научными дисциплинами. Давно ушли в прошлое те времена, когда основной задачей этой дисциплины считалась «ликвидация компьютерной безграмотности» и формирование «базовых навыков работы на компьютере» – у сегодняшних студентов-первокурсников, как правило, все в порядке и с компьютерной грамотностью, и с базовыми навыками.

Существует множество вузовских учебников и учебных пособий по информатике, рекомендованных студентам различных специальностей или групп специальностей. В этих учебниках информатика рассматривается как общеобразовательная дисциплина, задачами которой является формирование у студентов базовых представлений об информатике, вычислительной технике и машинной арифметике, а также практическое освоение студентами информационных технологий и пользовательских программных приложений, состав которых определяется профилем образовательной программы.

При всем многообразии учебников по информатике, ни один из них не может быть рекомендован студентам IT-специальностей в качестве основного, и это вполне объяснимо, так как для таких студентов профессиональное освоение информатики не может ограничиться какой-либо одной дисциплиной – различные аспекты этой науки и ее многочисленных приложений рассматриваются комплексом взаимосвязанных профессиональных дисциплин в течение всего периода обучения.

Вводный курс информатики для студентов IT-специальностей позиционируется как практическое введение в прикладные разделы этой науки, в котором рассматриваются ее базовые понятия, системы числового кодирования данных, типовая структура, принципы функционирования и алгоритмы взаимодействия аппаратных и программных компонентов простейшего персонального компьютера (файловая система, клавиатура, видеосистема).

Автор выражает благодарность Аркадию Андреевичу Медведеву, внимательно прочитавшему рукопись и сделавшему ряд профессиональных комментариев и замечаний методического характера, несомненно способствовавших повышению качества учебного пособия.

ИНФОРМАТИКА – ИСТОРИЯ СТАНОВЛЕНИЯ

Идея создания автоматического программно-управляемого вычислительного устройства была высказана английским математиком Чарльзом Бэббиджем в 1833 году, но только через 100 лет эти идеи нашли практическое воплощение. Основы построения вычислительных машин в их современном понимании были заложены английским математиком А.Тьюрингом и американцем Дж.Нейманом. А.Тьюринг сформулировал (1936 г.) понятие абстрактной вычислительной машины, получившей название «*машины Тьюринга*», которая, хотя и не была реализована в качестве действующего вычислительного устройства, до настоящего времени используется для моделирования алгоритмов и вычислительных процессов. Дж.Нейман предложил свою модель вычислительного устройства («*автомат Неймана*»), позволяющую, в отличие от машины Тьюринга, естественно моделировать параллельные вычислительные процессы.

Первым реально работавшим автоматическим вычислительным устройством считается электро-механический вычислитель Z1, собранный немецким инженером Конрадом Цузе в 1938 году. В 1944 году в США была создана расчетно-механическая машина Марк-1. В этих конструктивно сложных машинах в качестве элементной базы были использованы различные электромеханические устройства, что не позволило их разработчикам добиться желаемого уровня надежности работы этих машин и высокой производительности вычислений.

Новейшая история развития вычислительной техники связана с достижениями в области вычислительной математики, промышленной электроники и схемотехники. Элементная база ЭВМ первого поколения – это электронные лампы. Первый ламповый триггер был создан российским ученым Бонч-Бруевичем в 1918 году, годом позднее аналогичная схема была продемонстрирована американцами Икклзом и Джорданом.

Проекты первых вычислительных машин на электронных радиолампах были разработаны в Германии и США – это проект Z2 Конрада Цузе и проект ENIAC Джона Моучли. Машина Z2 была введена в эксплуатацию в 1940 году и использовалась на одном из берлинских авиационных заводов для расчетов геометрических параметров авиационных бомб, а опытная эксплуатация ENIAC началась в 1944 году в одной из лабораторий Пенсильванского университета.

Среди отечественных разработок ЭВМ первого поколения можно отметить проекты МЭСМ (малая электронно-счетная машина) и БЭСМ-1 (1949 – 1952), ЭВМ «Стрела» (1953) и серия ЭВМ «Урал».

В середине 50-х годов на смену ламповым ЭВМ пришли вычислительные машины 2-го поколения, элементная база которых была построена на полупроводниковых приборах. Существенно снизилась энергоемкость вычислений, машины стали более компактными и быстродействующими, они стали оснащаться достаточно емкими (для своего времени) запоминающими устройствами – все это привело к тому, что ЭВМ превратилась из быстродействующего программируемого калькулятора в мощный инструмент для надежного хранения и высокопроизводительной обработки больших объемов информации.

В результате сформировалось новое научно-техническое направление, получившее свое название от французского *L'informatique* (на немецком – *Die Informatik*, на испанском – *La informatica*, на русском – *информатика*). Англоязычное название информатики – *The computer science* – подчеркивает важную роль вычислительной техники в информатике.

Информатика – это комплексное междисциплинарное направление, в рамках которого изучаются процессы получения, передачи, хранения и обработки информации средствами электронно-вычислительной техники и связи. В широком смысле информатика – это наука об информационных процессах и их организации в человеко-машинных (социо-технических) системах.

Информатика – относительно молодая, но динамично развивающаяся наука, интегрирующая и использующая передовые достижения многих фундаментальных и прикладных наук – таких, как *теория информации*, изучающая способы восприятия, преобразования и передачи информации; *теория автоматов*, изучающая специальный класс дискретных систем переработки информации; *теория программирования*, изучающая математические модели и алгоритмы нахождения решений и способы кодирования этих алгоритмов в форме, воспринимаемой вычислительной системой; *теория управления*, изучающая процессы управления объектами разных классов и принципы построения систем, осуществляющих целенаправленную переработку информации.

Информатика составляет основу информационных технологий (ИТ) – инженерных методов автоматизированной обработки и использования информации. Повышение вычислительной мощности ЭВМ, интенсивное развитие научной базы информатики и практики компьютерного программирования привели

к тому, что за относительно небольшой исторический период ИТ нашли достойное место среди других (более традиционных) инженерных отраслей и сегодня эффективно применяются в различных сферах деятельности человека, в том числе и весьма далеких от инженерии.

Вслед за сменой поколений вычислительной техники, ИТ прошли в своем развитии ряд этапов, на каждом из которых перед ними ставились определенные цели и вырабатывались технологические инструменты для их достижения.

Первый этап развития ИТ (50-е и начало 60-х годов) характеризуется дефицитом машинных ресурсов. Основная технологическая проблема в этот период – низкая скорость вычислений и высокая стоимость самих вычислителей. В 1953 году один из основателей теории информации Клод Шеннон писал: *«Наши ЭВМ выглядят, как ученые-схоласты – при вычислении длинной цепи арифметических действий они значительно обгоняют человека, когда же их пытаются использовать для выполнения неарифметических операций, они оказываются неуклюжими и неприспособленными для такой работы».*

Основной целью ИТ на этом этапе было повышение эффективности компьютерной обработки данных по легко формализуемым алгоритмам, и эта цель достигалась путем повышения быстродействия ЭВМ и оптимизации программ по критериям длины машинного кода и затрат оперативной памяти на программную реализацию вычислительных алгоритмов.

В 50-е годы были сделаны важные шаги в направлении перехода от аппаратно-арифметического подхода к программированию на низкоуровневых машинно-ориентированных языках к алгоритмическим языкам высокого уровня, что привело к повышению производительности разработки программ¹.

В этот период были разработаны четыре базовых высокоуровневых языка: FORTRAN-1, нацеленный на программирование математических задач, COBOL, эффективно работавший с текстами и записями и предназначенный для разработки бизнес-приложений и решения экономических задач, язык ALGOL-58, в

¹ Идея использования высокоуровневых команд управления вычислениями была высказана Конрадом Цузе, им же был разработан первый высокоуровневый язык программирования Планкалькюль (Plankalkül – «исчисление планов»), реализованный в проекте Z3 1941 года и опубликованный только в 1948 году. Этот язык был свободно-переносимым, то есть независимым от системы команд машины, включал операторы условных переходов и циклов, обеспечивал возможность работы с подпрограммами и многомерными массивами.

котором впервые реализован блочный (структурный) подход к построению программы, и язык функционального программирования LISP, ориентированный на обработку списковых структур данных и специализированный для решения задач не численного характера.

Возможно, главной чертой *второго этапа* развития ИТ (с середины 60-х до конца 80-х годов), следует считать появление микропроцессорной техники и, как следствие – создание мини-ЭВМ и массовое распространение персональных компьютеров. Вот краткая история «миниатюризации» компьютеров:

1969 год – компания Honeywell выпускает на рынок первый домашний компьютер;

1971 год – выпущен первый *микропроцессор* – 4-разрядный *Intel 4004*.

1973 год – первый ПК Хероха Alto с графическим интерфейсом и прообразом «рабочего стола»;

1974 год – выпущен первый 8-разрядный микропроцессор *Intel 8080* (архитектурный аналог которого выпускался в СССР в составе микропроцессорного комплекта *K580*);

1977 год – первый компьютер для массового пользователя – серийный моноблок Apple-2 с цветной графикой и встроенным Basic-интерпретатором;

1978 год – выпущен первый 16-разрядный микропроцессор *Intel 8086* (отечественный аналог – *микропроцессор K1810BM80*), который имел 20-разрядную адресную шину, что позволило в 16 раз увеличить размер адресного пространства (до 1 Мб). Архитектура этого микропроцессора, получившая обозначение **x86**, фактически стала стандартом на длительный период развития персональных компьютеров.

1982 год – выпущен 16-разрядный *микропроцессор Intel 286* с увеличенной до 24-х разрядов адресной шиной, что позволило еще в 16 раз расширить емкость адресуемой памяти;

1983 год – IBM PC XT с увеличенным объемом оперативной памяти и жестким диском емкостью 10 М(!)б;

1984 год – первый серийно выпускаемый персональный компьютер АГАТ с 8-разрядным процессором отечественного производства;

1985 год – начало выпуска 32-разрядных микропроцессоров (*Intel 386 SX* и последующие модификации этой модели *386 DX* с интегрированным арифметическим сопроцессором, а также последующими моделями *Intel 486*, *Intel Pentium* и их модификациями.)

1986 год – первый ноутбук IBM PC Convertible с блоком питания от батареи, жестким диском и двумя 3-дюймовыми дисководами.

Еще одна важная черта второго этапа развития информационных технологий – использование межкомпьютерных коммуникаций. Попытки создания компьютерных сетей предпринимались еще в начале 60-х годов, когда несколько терминалов, оснащенных собственными устройствами ввода-вывода, соединялись с большой ЭВМ, обслуживающей все эти терминалы.

Первые, близкие к современным, способы соединения компьютеров в сети были разработаны после появления мини-ЭВМ и персональных компьютеров. Началом эпохи компьютерных сетей считается 1969 год, когда в США была создана сеть ARPANET, объединившая по телефонному кабелю несколько компьютеров, установленных в четырех университетах.

В 1971 году к этой сети были подключены еще 12 терминалов, а еще через два года к ней подключились и иностранные организации из Норвегии и Великобритании – так ARPANET стала прообразом современной глобальной сети интернет.

С переходом на новую элементную базу компьютеры стали дешевле и производительнее, они стали оснащаться быстродействующими дисковыми накопителями большой емкости, что дало толчок к развитию сетевых технологий и технологий баз данных. В результате области применения компьютеров вышли далеко за рамки научных и инженерных расчетов, и вслед за развитием рынка персональных компьютеров появился и рынок компьютерных программ, ориентированных на массового потребителя, а стоимость программного обеспечения компьютера стала больше стоимости его аппаратных компонентов.

Вскоре спрос на рынке программного обеспечения начал существенно превышать предложение по причине высокой трудоемкости процесса программирования. Прогнозы конца 60-х годов показывали, что при сохранении достигнутых к тому времени технологий разработки программ уже к середине 90-х годов программированием должно было бы заниматься все население планеты. Создавшаяся ситуация была определена, как «кризис программирования».

На фоне радикальных изменений в аппаратуре компьютерных систем произошло смещение критериев оценки ИТ – от эффективности исполнения программного кода вычислительным устройством к эффективности процессов разработки программ и их сопровождения на стадии эксплуатации.

Основным результатом радикального пересмотра критериев качества ИТ стало развитие концепций модульного программирования и структурного подхода к проектированию и программированию сложных программных систем.

В 1968 году в профессиональный оборот был введен термин *software engineering* – инженерия программного обеспечения, рассматривающая программный продукт как технически сложное промышленное изделие, к которому применимо понятие «жизненного цикла», включающего программирование лишь как один из этапов процесса производства программного продукта.

Была выработана новая концепция операционных систем, обеспечивающих повышение эффективности труда программистов за счет отказа от примитивного «пакетного режима» выполнения и отладки программ. В качестве примера можно привести ОС UNIX, успешно функционировавшую на мини-ЭВМ PDP-11 с начала 70-х годов, а также PC DOS с развитым «командным» языком, устанавливаемая на IBM-совместимые персональные компьютеры.

Были разработаны новые версии языков программирования (ALGOL-69, FORTRAN-66/77), и созданы новые языки – Pascal (1970) и Си (1972), позволяющие программисту оперировать сложными структурами данных и поддерживающие концепции структурного и модульного программирования, за счет чего улучшалась «читабельность» программного кода и сокращалось время отладки и модификации программ. Был создан первый непроецедурный высокоуровневый язык программирования PROLOG, в основе которого – язык предикатов математической логики.

Управление большими объемами информации было выделено в отдельную отрасль ИТ, получившую название «технологии баз данных». В практику проектирования баз данных введено информационное моделирование бизнес-процессов предметной области, были разработаны первые проектные модели и системы графического моделирования. Введено понятие «модели данных», разработаны теоретические основы и специализированные языки управления данными, была выработана концепция СУБД – систем управления базами данных.

В 1968 году компания IBM выпустила на рынок первую СУБД IMS, поддерживающую иерархическую модель данных. В 1969 году Конференцией по языкам систем данных (Conference on Data Systems Languages) была разработана сетевая модель данных, получившая название «модели CODASYL». В 1970 году сотрудник компании IBM Э.Кодд предложил реляционную модель данных и детально разработал математическую основу этой модели – реляционную алгебру, а также теорию нормальных форм и технологию нормализации баз данных. К середине 70-х идеи Э.Кодда и его последователей были реализованы в первой реляционной СУБД System R, выпущенной компанией IBM.

Третий этап развития информационных технологий, начавшийся в 90-е годы и продолжающийся в настоящее время, характеризуется переходом к объектно-ориентированной методологии разработки программных систем, дальнейшим развитием сетевых технологий и распределенных систем хранения и обработки данных, а также с получением практически значимых результатов в сфере искусственного интеллекта и *Data Mining* – мульти-дисциплинарной области, в рамках которой рассматриваются методы и компьютерные технологии извлечения ранее неизвестных знаний из больших объемов слабоструктурированных данных.

Достижения второго этапа развития ИТ позволили (как позже выяснилось – временно) преодолеть кризис программирования, но дальнейшее расширение сферы применения компьютерной техники в условиях динамично развивающегося рынка программного обеспечения потребовало повышения мобильности процессов разработки: стала объективной реальностью ситуация, когда заказчик программного продукта вносит *изменения в требования* к его функционированию не только на стадии эксплуатации (с чем разработчики уже успели смириться), но и *на стадиях проектирования и программирования*.

В этих условиях разработка программной системы превращается в ее постоянное перепроектирование и перепрограммирование, и потребовалась методология, обеспечивающая возможность оперативного внесения изменений в программу без существенных изменений ранее написанного кода. В результате была предложена *объектно-ориентированная* методология разработки программных систем, которая, используя достижения структурного подхода, радикально меняет как технологию проектирования системы, так и структуру программного кода ее компонентов.

Согласно структурному подходу решение задачи представляется в виде последовательности вызовов функций, реализующих определенные алгоритмы. Объектно-ориентированная архитектура программной системы представляется множеством взаимосвязанных объектов, взаимодействующих друг с другом путем передачи сообщений и/или предоставления ресурсов. Состояние программной системы определяется свойствами (значениями атрибутов) всех ее объектов, а функционирование системы рассматривается как процесс смены ее состояний в результате взаимодействия объектов.

Внедрение объектно-ориентированной методологии потребовало разработки соответствующего этой методологии языка графического моделирования, и такой язык был создан – он получил название UML (Unified Modeling Language). Были разработаны также поддерживающие этот язык CASE-средства (Computer Aided Soft Engineering), обеспечивающие процессы разработки и документирования программных проектов.

На смену языкам структурного программирования пришли объектно-ориентированные языки, наиболее известные из которых – C++, C#, Visual Basic, Java, Python.

Еще одна особенность ИТ рассматриваемого периода связана с прогрессом в области сетевых технологий и массовых телекоммуникаций: высокоскоростные (в том числе и беспроводные) каналы передачи данных, Web-технологии доступа к информационным ресурсам, облачные сервисы и распределенные хранилища данных, мобильная связь, IP-телефония и мультимедиа, мессенджеры и социальные сети – благодаря этим новым возможностям области применения ИТ существенно расширились, и самая разнородная информация стала массово доступной.

Доступность информационных ресурсов создала и новые угрозы – потребовалось защищать информацию от несанкционированного доступа со стороны пользователей, а также защищать самих пользователей от доступа к не предназначенной им информации. В этот период сформировалась отдельная отрасль ИТ – информационная безопасность, в рамках которой рассматриваются программно-технические, нормативно-правовые и организационные аспекты защиты информации в компьютерных системах.

Характеризуя третий этап развития ИТ, нельзя не упомянуть и о системах искусственного интеллекта (ИИ, англ. *Artificial Intelligence, AI*). Теоретические основы и методологические подходы к созданию таких систем были выработаны еще в 60-х годах трудами философов и нейробиологов, специалистов в области математической логики, прикладной статистики и теории алгоритмов, но для их практической реализации требовались вычислительные системы высокой производительности, которые появились только к началу 90-х годов.

Несмотря на отсутствие общепризнанного определения искусственного интеллекта, существует понимание того, что искусственный интеллект – это способность искусственных, то есть созданных человеком программно-технических систем, автоматически выполнять те виды человеческой деятельности, которые традиционно считаются творческими (интеллектуальными).

Технологии ИИ интенсивно развиваются в двух основных направлениях: *семиотическое* направление занимается созданием *баз знаний* и *экспертных систем*, имитирующих психические процессы мышления, рассуждения, логического вывода, и *биологическое*, в котором разрабатываются алгоритмы эволюционных вычислений, искусственные *нейронные сети* и *биокомпьютеры*.

Оба эти направления нашли свое применение при создании систем ИИ в самых различных областях: робототехника и управление беспилотными транспортными средствами, распознавание текстов и графических образов, компью-

терные игры, обучающие образовательные системы и системы самообучения машин, системы поддержки принятия решений в маркетинге, бизнесе и в медицине, многомерная аналитическая обработка данных, поиск неявных закономерностей в больших массивах статистических данных и многие другие.

Приведенный выше краткий исторический очерк дает представление об информатике как о комплексном научно-техническом направлении со своей теоретической базой, аппаратным и программным обеспечением и множеством практических приложений. Прикладные разделы информатики создают методологическую основу информационных технологий и программной инженерии, занимающейся разработкой и эксплуатацией автоматизированных информационных, информационно-аналитических, информационно-управляющих и информационно-телекоммуникационных систем.

Вычислительная система – это неразрывный комплекс аппаратного и программного обеспечения, в котором аппаратные компоненты работают под управлением компьютерных программ, а программы выполняются на аппаратной платформе, обрабатывают данные, хранимые на аппаратных устройствах, и взаимодействуют друг с другом, реагируя на сигналы, поступающие от аппаратных компонентов. При этом программа остается практически бесполезной без реализующей ее аппаратуры, а аппаратура без управляющей программы – не более, чем мертвое (хотя и весьма дорогостоящее) «железо».

Изучение теоретической информатики, а также процессов функционирования компонентов электронно-вычислительных машин и системного программного обеспечения, выходят за рамки вводного курса информатики – теория информации, схемотехнические основы и архитектура ЭВМ, низкоуровневое программирование и функционирование операционных систем рассматриваются в соответствующих профильных дисциплинах, включенных в учебные планы практически всех ИТ-специальностей.

В учебном пособии рассмотрены основы двоичного кодирования информации, принципы функционирования ЭВМ и примеры реализации этих принципов в персональном компьютере простейшей магистральной архитектуры. Основное внимание уделено алгоритмам функционирования файловой системы, системы обмена данными с периферийным оборудованием и видеосистемы ПК, а также структурам данных, обслуживающим эти алгоритмы.

2.1 Понятие информации

Латинское понятие *informatio*, давшее свое «имя» информатике, как целому научно-техническому направлению, может быть переведено на русский язык, как «разъяснение», «изложение» или «осведомленность», при этом слова, обозначающие это понятие, существовали во многих естественных языках (как, например, слово «информация» в русском) задолго до появления информатики. Клод Шеннон ввел в употребление технический термин «информация» применительно к *теории передачи кодов*, получившей впоследствии название *теории информации*, но само понятие информации все еще остается интуитивным и получает разные толкования в различных сферах деятельности человека.

В широком смысле слова *информация* – это отображение (*информационная модель*) *реального мира*, в узком техническом смысле *информация* – это *сведения, являющиеся объектом хранения, передачи и преобразования*. Информация – это философская категория, и ее формальное определение не может быть дано в рамках конкретной науки, но каждому интуитивно понятно, что *информация* – это *сведения, совокупность знаний о каком-либо процессе, явлении или предмете*.

Существуют и другие неформальные определения этого понятия, например: *информация* – это *совокупность сведений, уменьшающих неопределенность в выборе различных возможностей*. В последнем определении подчеркивается тот факт, что информация – это не любые сведения об объекте, а только *полезные* сведения, уменьшающие неопределенность в процессе принятия какого-либо решения. Информация – это определенный набор свойств (параметров) реального объекта, и проявление этих свойств и их восприятие другими объектами становится возможным только в процессе информационного обмена между объектами при их взаимодействии.

В материальном мире существует два фундаментальных вида материального взаимодействия: вещественное и энергетическое, которые подчиняются законам сохранения и в этом смысле являются симметричными: сколько один объект отдал, столько же другие и получили. В отличие от материальных взаимодействий, информационное является *несимметричным* и не подчиняется законам сохранения – объекты-приемники получают переданную им информацию, но при этом объект-источник ее не теряет.

Естественно, что информационное взаимодействие сопровождается энергетическим и/или вещественным, обеспечивающим перенос информационных сигналов посредством изменения параметров какой-либо материальной среды (например, акустической или электромагнитной).

С учетом сказанного сформулируем следующее определение: *любое взаимодействие между объектами, в процессе которого один из них приобретает некоторую субстанцию, а другой ее не теряет, будем называть информационным взаимодействием, а передаваемую при этом субстанцию будем называть информацией.*

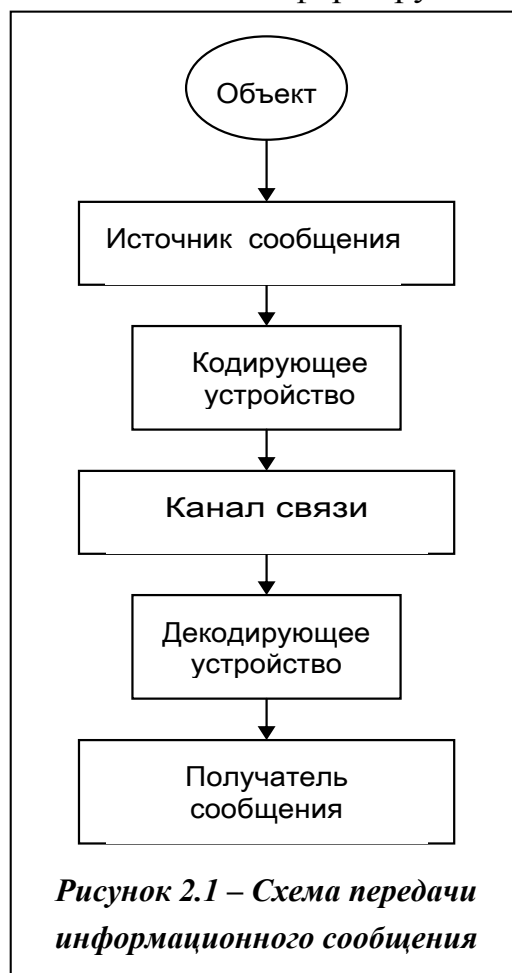
Информация представляется в форме *информационного сообщения*, передаваемого от источника сообщения к получателю по каналу связи (рисунок 2.1).

Источник сообщения получает информацию об объекте и формирует информационное сообщение, пользуясь некоторой системой знаков, которую будем называть *алфавитом*. Таким алфавитом может быть алфавит русского или английского языков, азбука Морзе, азбука Брайля, двоичный или десятичный цифровые алфавиты.

Кодирующее устройство производит преобразование формы представления информационного сообщения: при этом могут изменяться как исходный алфавит сообщения, так и физическая форма представления сообщения. На выходе кодирующего устройства сообщение представлено в форме сигнала, соответствующего параметрам канала связи.

Канал связи представляет собой некоторую физическую среду, которая изменяет свои энергетические параметры под воздействием передаваемого сигнала.

Декодирующее устройство регистрирует изменение этих параметров канала связи и преобразует принятые сигналы к физической форме и алфавиту, принятым у получателя сообщения. Таким образом, сообщение поступает к получателю в удобной для него форме, позволяющей получателю производить интерпретацию и обработку сообщения.



Перечислим основные факторы, присущие информационному взаимодействию.

1. Информационные взаимодействия возможны только в результате симметричных взаимодействий, связанных с передачей вещества и энергии. Формы вещества или энергии, с помощью которых переносится информация, называются *информационными кодами*.

2. Информационное взаимодействие может происходить только при взаимном соответствии свойств передающего и принимающего объектов. От свойств принимающего объекта зависит то, какую информацию он принимает, получая конкретный набор кодов. Комплекс свойств принимающего объекта, позволяющих ему воспринимать получаемые коды как некоторую информацию, называют *аппаратом интерпретации*.

3. *Информация, принимаемая объектом, является целесообразной для принимающего объекта, а нецелесообразная для этого объекта информация не будет им принята по причине отсутствия у него соответствующего аппарата интерпретации.*

4. *Информация, передаваемая объектом, является для него значимой, а процесс передачи информации является целенаправленным.*

2.2 Свойства информации

Информационные технологии – это способы целенаправленной обработки информации, реализация которых приводит к изменению ее свойств. Рассматривая свойства информации как объекта ИТ, выделяют два класса таких свойств: внутренние и внешние. Внутренние свойства информации – это свойства, органически присущие самому отображаемому объекту, а внешние ее свойства проявляются во взаимодействии объектов, участвующих в процессе информационного обмена.

Рассмотрим внешние свойства информации, оцениваемые получателем информационного сообщения, который, как правило, имеет некоторое представление о возможном его содержании и оценивает качество, полезность и возможность использования полученной информации в соответствии со своими ожиданиями.

При этом возможны следующие типовые ситуации:

а) полученная информация может соответствовать, не соответствовать или частично соответствовать запросу получателя;

б) информация может соответствовать его запросу, но ее оказалось недостаточно для решения соответствующей задачи;

в) информация оказалась недостоверной по причине наличия скрытых ошибок в информационном сообщении;

г) полученная информация несвоевременна (уже устарела или, наоборот, поступила преждевременно);

д) информация представлена в неудобной для получателя форме (например, использован незнакомый получателю алфавит);

е) информация недоступна (например, по причине аппаратной или программной несовместимости технических устройств или отсутствия прав доступа к ней у получателя сообщения).

В соответствии с рассмотренными ситуациями можно сформулировать следующие *внешние свойства информации*, проявляющиеся в процессе ее обработки получателем сообщения:

- **РЕЛЕВАНТНОСТЬ** – свойство информации, определяющее степень ее соответствия запросу получателя;
- **ДОСТОВЕРНОСТЬ** – свойство информации, определяющее наличие (и количество) скрытых ошибок;
- **ПОЛНОТА** – свойство информации исчерпывающее (для получателя) характеризовать отображаемый объект;
- **ЭРГОНОМИЧНОСТЬ** – свойство, характеризующее удобство формы представления информации в информационном сообщении;
- **ДОСТУПНОСТЬ** – свойство, характеризующее возможность получения информационного сообщения;
- **СВОЕВРЕМЕННОСТЬ** – свойство, характеризующее время получения информационного сообщения.

Если рассматривать другую группу свойств информации, связанных с взаимодействием источника информации с отображаемым объектом, то здесь важнейшим свойством информации является ее **АДЕКВАТНОСТЬ** – свойство информации однозначно соответствовать отображаемому объекту. С точки зрения получателя информационного сообщения адекватность является внутренним свойством информации, проявляющимся через такие ее внешние свойства, как релевантность, достоверность и полнота.

2.3 Количество информации

Реализация базовых процессов информационных технологий, обеспечивающих обработку, передачу или хранение информации, требует количественной оценки информационных сообщений, поэтому количество информации – важнейшее ее внутреннее свойство. Для количественной оценки информации необходимо выбрать единицы ее измерения, что позволит приписать определенной порции информации некоторое числовое значение.

Существует три основных метода оценки количества информации: алгоритмический, объемный и энтропийный.

Алгоритмический метод применяется в теории информации и теории алгоритмов. Согласно этому методу *количество информации, содержащееся в информационном сообщении, определяется алгоритмической сложностью компьютерной программы, воспроизводящей это сообщение*. Например, сообщение вида «0000» будет содержать меньше информации, чем сообщение вида «0101», так как программа, генерирующая первое сообщение, очевидно проще (короче) программы, генерирующей второе. Для того, чтобы реально оценить количество информации по алгоритмическому методу, необходимо задаться некоторым единым алгоритмическим языком, на котором следует записывать программы-генераторы оцениваемых информационных сообщений. Для этих целей используется язык *машины Тьюринга* – абстрактной модели простейшего вычислительного устройства, уже упоминавшейся во вводной части учебного пособия.

Объемный метод оценки количества информации – самый простой и очевидный: согласно этому методу, количество единиц информации, содержащихся в информационном сообщении, равно длине (количеству символов) самого сообщения. При всей простоте такого метода оценки он оказывается чувствительным к форме записи (алфавиту) сообщения. Следующий пример иллюстрирует тот факт, что при использовании объемного подхода одно и то же текстовое сообщение, записанное в разных алфавитах, будет иметь различную количественную оценку: «21» – 2 единицы, «XXI» – 3 единицы, «twenty one» – 10 единиц, «двадцать один» – 13 единиц.

Энтропийный метод оценки количества информации, принятый в теории информации и кодирования, использует следующую модель:

1). Получатель сообщения имеет определенное представление о его возможном содержании – это представление выражается вероятностями, с которыми он ожидает тот или иной вариант сообщения.

2). Общая мера неопределенности (энтропия) характеризуется некоторой математической зависимостью от совокупности этих вероятностей.

3). Количество информации, содержащейся в информационном сообщении, определяется тем, насколько уменьшится энтропия (мера неопределенности) после получения данного сообщения.

Рассмотрим простой пример. Из колоды игральных карт (32 карты) наугад выбирается одна карта. Всего имеется 32 равновероятных варианта получения информационного сообщения «выбрана конкретная карта», таким образом, общая мера неопределенности может быть оценена числом 32. После получения сообщения (выбора определенной карты) неопределенность полностью снимается, следовательно, число 32 определенным образом характеризует количество информации, содержащейся в полученном сообщении.

Очевидно, что чем больше вариантов сообщения, тем больше мера неопределенности и, соответственно, больше информации содержит полученное сообщение. Если бы в рассмотренном примере колода содержала не 32, а 256 различных игральных карт, то же самое сообщение «выбрана конкретная карта» содержало бы больше информации, чем в предыдущем случае.

Пусть N – количество равновероятных вариантов сообщения, а I – количество единиц информации, содержащейся в этом сообщении. В рассмотренных выше примерах предлагалась простейшая оценочная формула $I = N$, однако в теории информации принята другая (производная от рассмотренной выше) количественная оценка:

$$I = \log_2 N \dots\dots\dots (2.1)$$

Согласно этой оценке, количество информации, содержащейся в сообщениях «выбрана одна карта из возможных 32-х» и «выбрана одна карта из возможных 256» будет, соответственно, 5 и 8 единиц.

Обобщим приведенные рассуждения. Пусть информационное сообщение длиной X знаков задано на некотором P -ичном (P -арном) алфавите (то есть на алфавите, содержащем P различных знаков). Тогда количество различных вариантов такого сообщения определится по следующей формуле:

$$N = P^X \dots\dots\dots (2.2)$$

Подставив выражение (2.2) в выражение (2.1), получим формулу для оценки количества информации:

$$I = \log_2 N = \log_2 P^X = X \cdot \log_2 P \dots\dots\dots (2.3)$$

Как видно из формулы (2.3), количество информации пропорционально длине сообщения X , а коэффициент пропорциональности логарифмически возрастает с увеличением арности используемого алфавита. Таким образом, чем «богаче» алфавит, тем больше информации будет содержать сообщение фиксированной длины, заданное на этом алфавите.

Учитывая тот факт, что количество информации, содержащейся в сообщении любой длины, заданном на унарном алфавите ($P = 1$), тождественно равно нулю ($I = X \cdot \log_2 1 \equiv 0$), самым простым является двоичный ($P = 2$) алфавит, для которого формула (1.3) существенно упрощается:

$$I = X \cdot \log_2 2 = X \dots\dots\dots (2.4)$$

Как видно из формулы (2.4), количество информации, содержащейся в сообщении, заданном на двоичном алфавите, равно длине этого сообщения (отметим, что в случае с двоичным алфавитом энтропийный метод оценки сводится, по существу, к рассмотренному ранее объемному методу).

За единицу измерения информации принято количество информации, содержащееся в самом коротком сообщении ($X = 1$), заданном на самом простом (двоичном, $P = 2$) алфавите:

$$I = 1 \cdot \log_2 2 = 1 \dots\dots\dots (2.5)$$

Эта единица получила название **bit** – сокращение от двух английских слов: **binary digit** – двоичная цифра, или двоичный разряд².

Для оценки размеров информационных массивов, хранимых в запоминающих устройствах компьютеров, а также для оценки информационной емкости самих этих устройств, используется другая (техническая) единица количества информации, называемая *байтом*.

Байт (byte) - это минимальный размер ячейки памяти компьютера, имеющей уникальный адрес. В современных компьютерах 1 byte = 8 bit.

Соответственно, используются и более крупные единицы, производные от байта: килобайт, мегабайт, гигабайт и терабайт:

$$\begin{aligned} 1 \text{ Кб} &= 2^{10} \text{ б} = 1024 \text{ б} ; 1 \text{ Мб} = 2^{10} \text{ Кб} = 2^{20} \text{ б} = 1\,048\,576 \text{ б} \\ 1 \text{ Гб} &= 2^{10} \text{ Мб} = 2^{20} \text{ Кб} = 2^{30} \text{ б} = 1\,073\,741\,821 \text{ б} \\ 1 \text{ Тб} &= 2^{10} \text{ Гб} = 2^{20} \text{ Мб} = 2^{30} \text{ Кб} = 2^{40} \text{ б} = 1\,099\,511\,627\,776 \text{ б} \end{aligned}$$

² Существует и «настоящее» английское слово **bit**, переводимое на русский язык, как «частица, малое количество чего-либо» – очевидно, этот факт также был принят во внимание при выборе наименования единицы измерения информации.

В 1998 году Международной электротехнической комиссией (МЭК) было предложено иное наименование производных от байта единиц, чтобы отличать принятый в СИ префикс «кило-» ($10^3 = 1000$) от принятого в информатике $2^{10} = 1024$. Согласно определению МЭК, единица, равная 2^{10} байт = 1024 байта, называется «кибибайтом» (КиБ, KiB): префикс «киби-» получен от слов «кило» и «бинарный». Соответственно, 2^{10} КиБ = 1 МиБ (мебибайт), 2^{10} МКиБ = 2^{20} КиБ = 2^{30} Б = 1 ГиБ (гибибайт) и т.д.

На практике применяются как традиционные наименования производных от байта единиц количества информации, так и наименования МЭК.

2.4 Информация, данные и знания – модель DIKW

В программировании и прикладной информатике широко используются такие популярные в профессиональной среде словосочетания, как *тип данных* (*data type*), *обработка данных* (*data processing*), *формат представления данных* (*data presentation format*), *база данных* (*data base*), *большие данные* (*big data*), *интеллектуальный анализ данных* (*data mining*), в которых термин *данные* заменяет близкий к нему по смыслу термин *информация*.

Данные – это набор разрозненных фактов, необработанный материал, который может стать источником некоторой информации. Используя термин *данные*, мы подчеркиваем техническую сторону *информации* и не затрагиваем ее содержательный и семантический (смысловой) аспекты.

Данные – это то, что может быть получено в результате измерений или выполнения логико-математических операций и при этом представлено в форме, пригодной для хранения и последующей обработки. Например, в программировании отнесение каких-либо *данных* к определенному *типу* предполагает определенный *способ хранения* этих данных в памяти компьютера и определенный набор допустимых *операций* над этими данными.

«Данные» – это еще не «информация», информацию из данных надо еще извлечь, применяя для этого определенные методы аппарата интерпретации. Одни и те же данные могут быть источником самой различной, в том числе и противоречивой, информации в зависимости от того, какие методы были применения для ее извлечения. Несоответствие данных методам, применяемым для их обработки, или же низкое качество самих этих методов, могут приводить и к тому, что объективно корректные данные могут стать источником недостоверной информации.

В качестве примера можно привести низкую достоверность информации прогнозного характера, полученной при обработке выборки экспериментальных данных не вполне адекватным методом, что в результате привело к большой погрешности прогноза – либо по причине малого объема этой выборки, либо потому, что метод не учитывал наличия в выборке случайных выбросов.

Данные по своей природе могут быть как объективными – например, температура и влажность воздуха, атмосферное давление, направление и сила ветра, полученные с соответствующих метео-датчиков, так и субъективными – например, оценки в школьном журнале, выставленные учителем. Информация же всегда субъективна, так как она получена путем обработки данных кем-то разработанными (то есть субъективными) алгоритмами интерпретации.

Информация формируется в результате анализа и выявления соотношений и взаимосвязей между разрозненными фактами (элементами «данных») и позволяет описать соответствующие события, процессы или явления, то есть дать ответы на вопросы типа «Что?», «Кто?», «Где?», «Когда?», «Сколько?» и «Почему?». Например, анализ метеоданных за длительный период времени может позволить выявить зависимости между направлением ветра, атмосферным давлением и температурой воздуха, а сравнительный анализ данных школьного журнала позволит сформировать рейтинговые списки учеников, учителей, учебных предметов или их отдельных тем.

Информация, полученная в результате интерпретации данных, может послужить основанием для принятия какого-либо решения только в том случае, если она будет преобразована в соответствующие знания.

Знания – это совокупность фактов, выявленных закономерностей и правил, с помощью которых может быть решена поставленная задача. Знания получаются в результате синтеза полученной информации и человеческого разума, они формируются в процессе восприятия и усвоения результатов анализа разнородной взаимосвязанной информации, полученной из множества источников данных, с учетом практического опыта людей, их способностей, интуиции, убежденности и мотиваций.

Отметим основные свойства знаний, отличающие их от информации:

- *структурированность* – знания должны быть представлены множеством *взаимосвязанных* компонентов с целью обеспечения их *эффективного усвоения* человеком (понять, запомнить или вспомнить забытое) или *эффективного доступа* к ним компьютерными средствами;

- *непротиворечивость* – знания не должны противоречить друг другу;
- *лаконичность* – знания должны быть минимально избыточными и не должны быть зашумленными, что позволяет быстро их осваивать и перерабатывать, повышая «коэффициент полезного содержания»;
- *процедурность* – знания нужны для того, чтобы их использовать, то есть применять к ним процедуры принятия решений, а также процедуры хранения, вывода и передачи знаний другим субъектам.

Существуют определенные трудности в понимании отличий и трактовке терминов *данные*, *информация* и *знания*, что вызвано их кажущейся синонимичностью. Даже известное в инженерии знаний направление *Data Mining* по-разному переводится на русский язык, причем в различных переводах используются все три этих понятия: «интеллектуальный анализ *данных*», «извлечение *информации*» или «добыча *знаний*».

Несмотря на отмеченные различия, все эти понятия тесно связаны и являются элементами единого процесса обработки данных, иллюстрируемого «информационной пирамидой» (рисунок 2.2) и известного как «Модель *DIKW*»: *Data – Information – Knowledge – Wisdom*.



Рисунок 2.2 – Информационная пирамида (модель *DIKW*)

В основании пирамиды находятся *данные (data)* – факты о реальном мире, полученные в результате наблюдений и измерений и сохраненные в форме, удобной для компьютерного хранения и последующей обработки.

Следующий уровень пирамиды занимает *информация (information)*, формируемая в процессе интерпретации данных путем выявления и осмысления взаимосвязей между их отдельными элементами. *Информация*, в отличие от данных, несет в себе некоторый смысл.

Еще более высокий уровень – это *знания (knowledge)*, которые получаются в результате целенаправленного восприятия и осмысления информации. Знания по своей природе процедурны и могут использоваться в процессах выработки и принятия решений.

Вершину информационной пирамиды занимают *глубокие знания, понимание или мудрость (wisdom)*. На этом этапе обработки данных к знанию добавляется понимание того, где и как можно использовать полученные знания, в том числе и за границами исследуемого процесса или явления.

На каждом новом уровне обрабатываемые данные становятся более структурированными, процедурными и пригодными для использования в процессе принятия решений. Если информация *описывает* исследуемый объект и дает ответы на вопросы «Что?», «Сколько?» и «Почему?», знание – позволяет выработать технологические *инструкции* и получить ответы на вопрос «Как?», то понимание дает нам *объяснение* и позволяет ответить на более сложные вопросы – «Зачем?», «Как лучше всего?».

Три нижних уровня пирамиды представляют прошлое и настоящее – *данные, информация* и *знания* позволяют описать существующие процессы, явления, связи, и только *понимание* на верхнем уровне дает возможность «предсказания будущего» – то есть позволяет сделать обоснованные прогнозы.

При движении вверх по информационной пирамиде *объемы данных* преобразуются в *ценность знаний*, однако гарантией высокой ценности знаний является не столько большой объем данных, сколько высокое качество методов их интерпретации, процедур обработки и использования информации.

Не всякие данные могут быть преобразованы в информацию, и не всякая информация сможет превратиться в знания. В качестве примера можно рассмотреть текст научной статьи, написанный на незнакомом читателю языке. С формальной точки зрения текст статьи – это данные, закодированные символами некоторого алфавита. При отсутствии аппарата интерпретации такие данные не смогут быть восприняты и осмыслены нашим читателем, то есть не смогут быть преобразованы в информацию. С помощью аппарата интерпретации (в данном примере – словаря или переводчика, знающего оба языка) можно сделать кор-

ректный перевод статьи, превратив тем самым данные в информацию, пригодную для восприятия. Теперь статью можно прочитать, однако не факт, что эта информация сможет получить статус знаний и в дальнейшем будет использована для принятия решений – возможно, читатель недостаточно профессионально подготовлен для осмысления полученной информации, или для понимания содержания статьи необходимо привлечение дополнительной информации.

2.5 Контрольные задания

Здание 2.1: Сформулируйте внешние свойства информации. Приведите примеры недостоверной, нерелевантной, неполной и неэргономичной информации, а также возможные причины недоступности информации.

Здание 2.2: *Автоматизированная система анализа пассажиропотоков*

Несколько конкурирующих компаний предоставляют услуги общественного транспорта жителям мегаполиса. Сформированы маршруты движения городских автобусов, в каждом автобусе установлены терминалы по продаже билетов (или другие устройства, регистрирующие факты оплаты поездок пассажирами), все эти терминалы связаны компьютерной сетью и сохраняют в базе данных информацию о каждой поездке (маршрут, координаты точки начала поездки, № автобуса, дата и время начала поездки, ее стоимость).

Транспортные компании используют информацию о выполненных поездках для решения оперативных задач финансовой отчетности, при этом информация за прошлые периоды оказывается невостребованной и хранится в архивах «на всякий случай».

За многие годы работы в базах данных транспортных компаний накопилось огромное количество исторической информации, и, наконец, «всякий случай» наступил – Департамент транспорта городской администрации принял решение о покупке у транспортных компаний этих баз данных для их использования при решении задач оптимизации пассажиропотоков.

Были подготовлены следующие аналитические отчеты на основе полученной информации:

- 1) Средняя загрузка каждого маршрута (поездки за сутки).
- 2) Сравнительный анализ средней загруженности маршрутов по временам года.
- 3) Сравнительный анализ средне-суточной загруженности маршрутов по дням недели.

- 4) Сравнительный анализ средне-часовой загруженности маршрутов по времени суток.

По результатам проведенного анализа Департаментом транспорта были приняты следующие решения:

- 1) Допустить еще одну транспортную компанию для увеличения количества автобусов на следующих маршрутах (список прилагается).
- 2) Увеличить количество автобусов на маршрутах (список прилагается) в дневное время по выходным и праздничным дням.
- 3) Проложить ветку метрополитена для связи центра города с районами (список районов прилагается), так как дальнейшее увеличение интенсивности движения наземного транспорта по этим маршрутам требует расширения улиц, а снос жилых домов обойдется городскому бюджету существенно дороже строительства метро.

Определить содержание информационной пирамиды (данные – информация – знания) для рассмотренных задач регистрации первичной информации о поездках пассажиров, подготовки аналитических отчетов и принятия решений.

Здание 2.3: Автоматизированная система маркетингового анализа в торговле

В торговом зале супермаркета установлены кассовые аппараты, объединенные в локальную сеть и обеспечивающие оперативную регистрацию в базе данных всех операций по продаже товаров покупателям супермаркета: категория товара, его наименование, цена и количество, дата и время покупки. Для постоянных клиентов (владельцев дисконтных карт) дополнительно указывается код клиента и текущий размер предоставленной ему торговой скидки.

В офисе супермаркета работают специалисты-маркетологи, формирующие следующие *аналитические отчеты* по результатам выполнения соответствующих запросов к базе данных торгового учета:

- 1) Текущие складские запасы (по категориям товаров).
- 2) Объемы продаж за период времени (сутки, неделя, месяц, квартал, год) в денежном исчислении.
- 3) Сравнительный анализ объемов продаж по категориям товаров.
- 4) Почасовой анализ динамики продаж в течение суток.
- 5) Средняя доля продаж постоянным покупателям.

Управляющий супермаркетом анализирует содержание представленных ему отчетов и, после дополнительных консультаций с маркетологами и экономистами, принимает следующие решения:

- 1) Пополнить складские запасы интенсивно продаваемых товаров.
- 2) Установить 50-процентную скидку на цены товаров, срок реализации которых приближается к критическому.
- 3) Объявить акцию «У нас – всегда низкие цены!» (постоянная скидка 0,5% на все товары ограниченного перечня категорий).
- 4) Объявить акцию: «Кто ходит в гости по утрам, тот экономит умно!» (скидка 10% на все товары, купленные до 8-00);
- 5) отменить дисконтные карты постоянным покупателям.

Определить содержание информационной пирамиды (данные – информация – знания) для рассмотренных задач оперативного учета продаж, маркетингового анализа и принятия перечисленных выше решений.

Здание 2.4: Автоматизированная система анализа успеваемости студентов

10 групп студентов-первокурсников Института Компьютерных Наук сдали экзамены по шести дисциплинам: математике, информатике, программированию, астрологии, хиромантии и основам научного оккультизма. Преподаватели зафиксировали в базе данных результаты сдачи экзаменов каждым из студентов по каждой дисциплине (оценки по 100-балльной шкале).

По окончании экзаменационной сессии для Директора Института были подготовлены печатные аналитические отчеты трех видов:

- 1) Групповые экзаменационные ведомости по каждой из дисциплин.
- 2) Рейтинговые списки студентов, ранжированные по среднему баллу студента по всем дисциплинам.
- 3) Рейтинговые списки студенческих групп, ранжированные по среднему баллу всех студентов группы по всем дисциплинам.

По результатам анализа содержимого отчетов Директор Института принял следующие решения, оформленные соответствующими приказами:

- 1) Исключить экзаменационные ведомости из перечня предоставляемых Директору отчетов по причине их низкой информативности и большого объема неструктурированной информации, что затрудняет процесс принятия решений.
- 2) Назначить следующим студентам (список прилагается) повышенный размер стипендии на следующий семестр за отличную успеваемость по всем дисциплинам.

- 3) Поощрить кураторов следующих групп (списки номеров групп и фамилии кураторов прилагаются) за высокие показатели академической успеваемости групп.
- 4) Провести дополнительное исследование причин плохой успеваемости студентов по информатике, математике и программированию по сравнению с их высокими достижениями в освоении астрологии, хиромантии и основам научного оккультизма.

Определить:

- 1). Содержание элементов информационной пирамиды (данные – информация – знания), необходимых для решения рассмотренных задач оперативного учета и анализа успеваемости студентов Института Компьютерных Наук, а также для принятия Директором Института решений №1, 2 и 3.
- 2). Состав дополнительных данных, информации и знаний, необходимых для реализации 4-го решения Директора Института.

ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В КОМПЬЮТЕРНЫХ СИСТЕМАХ**3.1 Числовое кодирование данных**

С точки зрения пользователя, не обремененного глубокими знаниями в области информатики, современный компьютер – это универсальный гаджет, на который можно «скачать и установить» множество различных «приложений», которые помогут ему (пользователю) писать, читать и рисовать, воспроизводить, обрабатывать и записывать аудио-, фото- и видеоматериалы, обеспечивать телефонную и видео-связь, осуществлять поиск информации в глобальной сети и, разумеется, играть в разнообразные игры. Длина этого списка возможностей с каждым годом увеличивается, и где-то в конце списка пользователь, конечно, упомянет и возможность считать (зачем-то же нужен калькулятор или MS Excel ?).

Не будем разочаровывать не слишком продвинутого пользователя, пусть он и дальше считает, что компьютер – это и писатель, и художник, и музыкант, и т.д., хотя на самом деле компьютер – это, в первую очередь, *computer*, то есть *вычислитель*, и главная его возможность – это возможность выполнять арифметико-логическую обработку чисел.

В соответствии с одним из принципов Фон Неймана (которые нам предстоит детально рассмотреть в следующей главе учебного пособия), любая информация (если более строго, то не информация, а данные) должна быть представлена в памяти компьютера в числовом двоичном коде, а процесс обработки этой информации компьютерными программами (также представленными в виде двоично-закодированных машинных команд) сводится к вычислительным процедурам, реализуемым аппаратными компонентами компьютера.

Так, например, при редактировании текстового файла ASCII-формата для преобразования заглавной буквы в строчную надо всего лишь прибавить 32 к числовому коду этой заглавной буквы, а процедура сортировки текстовых строк (расстановки их в алфавитном порядке) сводится к операции сравнения значений числовых кодов символов текста.

Другой пример: для того, чтобы перекрасить текстовый символ, изображенный на экране монитора, из синего цвета в зеленый, достаточно прибавить единицу к числовому коду атрибута этого символа и записать измененный код в соответствующую ячейку видеопамати компьютера. Чтобы перекрасить фон, на котором изображен текстовый символ, из синего в зеленый цвет, к числовому коду атрибута этого символа надо прибавить не единицу, а число 16.

Разумеется, для того, чтобы корректно обрабатывать информацию, надо хорошо понимать систему ее числового кодирования (то есть преобразования *информации в данные*) и представления данных в памяти компьютера.

Аппаратную основу электронных вычислительных устройств составляет двустабильный логический элемент – *триггер*, выходной сигнал которого может принимать одно из двух устойчивых состояний, одно из которых ассоциируется с числом 0, а другое – с числом 1. Вследствие этого, двоичный алфавит является естественным алфавитом для хранения и обработки информации в вычислительных устройствах, а двоичная система счисления положена в основу машинной арифметики и используется в качестве базовой системы счисления для технической реализации вычислительных процессов.

Двоичная запись числа является крайне неэкономичной, поэтому для записи двоичных чисел часто используют шестнадцатеричную систему счисления, в которой запись числа получается в четыре раза короче, а перевод чисел из двоичной системы в шестнадцатеричную и обратно производится по очень простому алгоритму, удобному даже для устного счета.

3.2 Системы счисления

Определение 1. **Система счисления** – это множество правил записи и именования чисел и правил выполнения базовых арифметических операций с этими числами.

Определение 2. **Базисные числа.** В любой системе счисления определен ограниченный набор базисных чисел, таких, что операции над ними позволяют вычислить значение любого другого числа в данной системе счисления.

Определение 3: **Цифра** – это знак (графический символ), используемый для записи базисного числа в некоторой системе счисления. В соответствии с данным определением, количество цифр и количество базисных чисел одной системы счисления должны совпадать.

Цифровая запись числа в определенной системе счисления однозначно определяет его значение, однако способы вычисления значений чисел, записанных в системах счисления разных типов, также будут различными.

3.2.1 Аддитивные системы счисления

В **аддитивных** системах счисления (от англ. *add* – прибавить, сложить) значение числа определяется суммированием и/или вычитанием базисных чисел, представленных в записи этого числа соответствующими цифрами. Классические примеры – унарная и римская системы счисления.

В *унарной* системе счисления, хорошо знакомой каждому под названием «счетные палочки», определено единственное базисное число «один» и соответствующая ему цифра «единица», обозначаемая символом «1». Значение десятичного числа 8 в унарной системе будет записано как 1111111; операция сложения $2 + 3 = 5$ – как $11 + 111 = 11111$; операция вычитания $5 - 4 = 1$ – как $11111 - 1111 = 1$; операция умножения $2 \times 3 = 6$ – как $11 + 11 + 11 = 11111$ или как $111 + 111 = 11111$. Такая система счисления очень легка в освоении, алгоритмы выполнения арифметических операций – просты и очевидны, что следует даже из их наименований ("прибавить", "отнять", "умножить"). Очевиден также и существенный недостаток унарной системы счисления – она непригодна для оперирования с большими числами.

Римская система счисления.

Другая аддитивная система счисления, получившая название «римской», обеспечивает более компактную (по сравнению с унарной системой) запись чисел, так как использует большее количество базисных чисел и, соответственно, большее количество цифр для их обозначения. В качестве цифр система использует буквы латинского алфавита: цифра "I" обозначает базисное число "один", V – пять, X – десять, L – пятьдесят, C – сто, D – пятьсот, M – тысяча. При записи чисел в римской системе счисления значением числа является алгебраическая сумма базисных чисел, представленных в записи числа соответствующими цифрами. При этом не любая последовательность цифр в записи числа является корректной, например:

- не разрешается записывать рядом более трех одинаковых цифр;
- если три левые крайние цифры в записи числа не одинаковы, то крайняя левая цифра должна иметь большее значение, чем соседняя с ней правая цифра;
- цифры записывают, как правило, в порядке убывания значений базисных чисел слева направо.

Если число записано корректно, то вычисление его значения производится по следующему алгоритму (таблица 3.1):

1. Начальное значение числа полагают равным нулю. Начинают последовательную обработку записи числа с крайней левой цифры.
2. Если очередная обработанная цифра - последняя (крайняя правая) в записи числа, то:
 - если вычисленное ранее текущее значение числа равно нулю, значение числа полагают равным значению очередной обработанной

- цифры (в этом случае эта цифра – единственная в записи числа) и работа алгоритма заканчивается;
- в противном случае вычисленное ранее текущее значение числа считается окончательным результатом и заканчивается работа алгоритма;
3. В противном случае сравнивают значения текущей цифры и соседней с ней правой цифры.
 4. Если значения цифр в паре одинаковы, вычисляют новое текущее значение числа как сумму предыдущего текущего значения и значения правой цифры в паре, и далее переходят к п.2 алгоритма (другими словами, значения всех одинаковых соседних цифр суммируются).
 5. Если значение очередной (правой) цифры не совпадает со значением левой (уже обработанной) цифры, то:
 - если очередная цифра – последняя (крайняя правая) в записи числа, ее значение суммируется с накопленным текущим значением числа, и заканчивается работа алгоритма;
 - в противном случае очередная цифра вместе с соседней с ней правой цифрой образуют новую пару.
 6. В случае, когда левая цифра пары представляет меньшее базисное число, чем правая, ее значение вычитается из значения правой цифры, в противном случае – оба значения складываются, и результат вычисления суммируется с предыдущим текущим значением числа;
 7. Далее выполняется п. 2 алгоритма.

Таблица 3.1 – Примеры «римской» записи десятичных чисел

Запись числа		Алгоритм вычисления десятичного значения числа
Римская система	Десятичная система	
C	100	[100]
IV	4	[5 – 1]
	264	[100 + 100] + [50 + 10] + [5 – 1]
CCCXCVII	397	[100 + 100 + 100] + [100 – 10] + [5 + 1] + [1]
DCXLIX	649	[500] + [100] + [50 – 10] + [10 – 1]
MCCVII	1207	[1000 + 100] + [100 + 5] + [1 + 1]
IV	Запись является некорректной, так как не выполняется ограничение №2, корректная запись – III ([1 + 1 + 1])	
CCLXIV XXXX	Запись является некорректной, так как не выполняется ограничение №1, корректная запись – XL ([50 – 10])	

Недостатком римской системы является отсутствие строго формализованных правил записи и интерпретации чисел, что затрудняет выполнение арифметических действий с многозначными числами. Например, число 649 можно записать двумя способами, не нарушая установленных правил записи: и как DCXLIX ($[500]+[100]+[50-10]+[10-1]$), и как DCIL ($[500]+[100]+[50-1]$).

3.2.2 Позиционные системы счисления

В позиционных системах счисления, называемых также *аддитивно-мультипликативными*, (от англ. *Add* – прибавить, сложить, *Multiply* – умножить) значение числа определяется комбинацией операций сложения и умножения базисных чисел, представленных в записи этого числа соответствующими цифрами.

Классические примеры – десятичная, двоичная, восьмеричная и шестнадцатеричная системы счисления. Все эти системы являются *позиционными*, так как вклад каждого базисного числа, представленного в записи числа соответствующей цифрой, определяется не только его значением, но и номером позиции (разряда), в которой записана эта цифра.

3.2.2.1 Десятичная система счисления

Формула (3.1) иллюстрирует алгоритм вычисления значения десятичного числа N_{10} по его цифровой записи. Значение числа получается суммированием произведений базисных чисел, представленных в записи этого числа соответствующими цифрами, на их позиционные веса.

$$N_{10} = \sum (b_i \times 10^i), \quad (3.1)$$

где:

i – номер позиции, в которой записана цифра, представляющая базисное число. Позиции записи целой части числа нумеруются справа налево, начиная от разряда единиц ($i = 0$), затем – десятков ($i = 1$) и т.д. до $i = n - 1$, где n – количество разрядов в целой части числа. Позиции записи дробной части числа нумеруются слева направо, начиная от разряда десятых ($i = -1$), затем – сотых ($i = -2$) и т.д. до $i = -m$, где m – количество разрядов в дробной части числа.

b_i – значение базисного числа (от 0 до 9), записанное в i -той позиции;

10 – основание десятичной системы счисления;

10^i – «позиционный вес» – числовое значение каждой единицы базисного числа, записанного в i -той позиции.

Рассмотрим пример записи и интерпретации значения числа 1234,56 в десятичной системе счисления.

$$1234,56 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1} + 6 \times 10^{-2}$$

Число 10 является *основанием* десятичной системы счисления (от которого эта система и получила свое название):

- в десятичной системе каждые 10 единиц младшего разряда объединяются в одну единицу соседнего старшего разряда: 10 единиц = 1 десяток, 10 десятков = 1 сотня и т.д.;
- в десятичной системе определено 10 базисных чисел: «ноль», «один», «два», ... «девять»;
- в десятичной системе используется 10 цифр (графических символов «0», «1», «2», ... «9») для записи соответствующих базисных чисел.

Все сказанное выше для десятичной системы можно распространить и на любую другую позиционную k -ичную систему счисления:

$$N_k = \sum (b_i \times k^i), \quad (3.2)$$

- основание системы счисления равно числу k ;
- k единиц младшего разряда числа объединяются в одну единицу старшего разряда;
- определено ровно k базисных чисел со значениями от 0 до $k-1$;
- для записи базисных чисел используется ровно k цифр;
- позиционный вес p_i для i -той позиции ($n-1 \geq i \geq -m$) вычисляется как k^i , где n и m – количество разрядов в целой и дробной частях k -ичной записи числа;
- значение числа по его записи вычисляется, как сумма произведений базисных чисел, представленных в записи числа соответствующими цифрами, на их позиционные веса.

3.2.2.2 Двоичная система счисления

Основание двоичной системы – число 2 (заметим, что это число здесь записано не в двоичной, а в привычной нам десятичной системе счисления).

Базисные числа в этой системе счисления – число «ноль» и число «один», обозначаемые цифрами 0 и 1 в записи двоичных чисел. При поразрядном сложении 2 единицы младшего разряда объединяются в одну единицу старшего (действуют все правила сложения/вычитания «в столбик» – перенос единицы в старший разряд при сложении или заем из старшего разряда при вычитании):

$$\begin{aligned}
1_2 + 1_2 &= 10_2; & 1_2 + 1_2 + 1_2 &= 11_2; \\
11_2 + 10_2 &= 101_2; & 111_2 + 111_2 &= 1110_2; \\
101_2 - 11_2 &= 10_2; & 111_2 + 1110000_2 &= 1110111_2
\end{aligned}$$

Для вычисления десятичного значения двоичного числа применяется описанная ранее общая формула, например:

$$\begin{aligned}
100110011,101_2 &= \\
1 \times 2^8 + 0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} &= \\
256 + 32 + 16 + 2 + 1 + 0,5 + 0,125 &= 307,625_{10}.
\end{aligned}$$

Полезное замечание: для двоичной системы счисления расчетную формулу для определения числового значения можно упростить – суммированию подлежат позиционные веса только тех позиций, в которых записана единица, а остальные позиции – игнорируются.

Заметим, что все вычисления производились в десятичной системе счисления, и полученный результат вычислений – десятичное число. Следовательно, рассмотренная выше формула может быть положена в основу алгоритма перевода чисел из двоичной системы счисления в десятичную и обратно.

Пример: переведем число $1234,125_{10}$ в двоичную систему счисления. Для этого представим исходное число в виде суммы слагаемых – максимально возможных целочисленных степеней двойки, начиная с наибольшего:

$$1234,125 = 1024(2^{10}) + 128(2^7) + 64(2^6) + 16(2^4) + 2(2^1) + 0,125(2^{-3}),$$

и затем сформируем двоичную запись числа, записав единицы в десятом, седьмом, шестом, четвертом, третьем, первом и «минус третьем» разрядах и нули – во всех остальных промежуточных разрядах. В результате получим:

$$1234,125 = 10011010001,001_2.$$

3.2.2.3 Восьмеричная система счисления

Основание восьмеричной системы – число 8.

В этой системе счисления восемь базисных чисел: ноль, один, два, три, четыре, пять, шесть и семь, и восемь цифр, заимствованных из десятичного алфавита: 0, 1, 2, ..., 7.

При поразрядном сложении 8 единиц младшего разряда объединяются в одну единицу старшего, при поразрядном вычитании заемная единица старшего разряда рассматривается как 8 единиц младшего:

$$\begin{aligned}
1_8 + 2_8 &= 3_8; & 3_8 + 4_8 &= 7_8; & 4_8 + 4_8 &= 10_8; \\
4_8 + 5_8 &= 11_8; & 23_8 + 56_8 &= 101_8; & 56_8 - 7_8 &= 47_8.
\end{aligned}$$

Для вычисления десятичного значения восьмеричного числа (перевод из восьмеричной системы счисления в десятичную) применяется описанная ранее общая для всех позиционных систем формула:

$$\begin{aligned}
 1234,56_8 &= \\
 1 \times 8^3 + 2 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 + 5 \times 8^{-1} + 6 \times 8^{-2} &= \\
 1 \times 512 + 2 \times 64 + 3 \times 8 + 4 \times 1 + 5/8 + 6/64 &= \\
 512 + 128 + 24 + 4 + 0,625 + 0,09375 &= 668,71875_{10}.
 \end{aligned}$$

Для обратного перевода из десятичной в восьмеричную систему надо представить исходное десятичное число в виде суммы произведений базисных чисел восьмеричной системы счисления на целочисленные степени восьмерки и затем записать эти числа в соответствующих позициях. Например:

$$58,125_{10} = 7 \times 8^1 + 2 \times 8^0 + 1 \times 8^{-1},$$

следовательно, в первой позиции восьмеричного числа будет записана цифра 7, в нулевой – цифра 2, а в «минус первой» позиции – цифра 1:

$$58,125_{10} = 72,1_8.$$

Для перевода из двоичной системы счисления в восьмеричную можно использовать уже знакомую формулу, однако при этом следует нумеровать позиции и выполнять все вычисления в восьмеричной системе счисления:

$$\begin{aligned}
 100110011,101_2 &= 2^{10} + 2^5 + 2^4 + 2^1 + 2^0 + 2^{-1} + 2^{-3} = \\
 400_8 + 40_8 + 20_8 + 2_8 + 1_8 + 4/10_8 + 1/10_8 &= 463,5_8
 \end{aligned}$$

3.2.2.4 Шестнадцатеричная система счисления

Основание шестнадцатеричной системы – число 16 (записанное здесь в десятичной системе счисления).

В этой системе шестнадцать базисных чисел: *ноль, один, два, три, четыре, ..., девять, десять, одиннадцать, ... , пятнадцать* (также записанных здесь в десятичной системе), и шестнадцать цифр, первые десять из которых заимствованы из десятичного цифрового алфавита: 0, 1, 2, 3, ..., 9, а недостающие шесть – из латинского буквенного: A, B, C, D, E и F.

При поразрядном сложении 16 единиц младшего разряда объединяются в одну единицу старшего, при поразрядном вычитании «заемная» единица старшего разряда рассматривается как 16 единиц младшего:

$$\begin{aligned}
 1_{16} + 2_{16} &= 3_{16}; & 8_{16} + 2_{16} &= A_{16}; & B_{16} + 2_{16} &= D_{16}; \\
 E_{16} + 2_{16} &= 10_{16}; & F_{16} + F_{16} &= 1E_{16}; & 20_{16} - 1_{16} &= 1F_{16}
 \end{aligned}$$

Для вычисления десятичного значения шестнадцатеричного числа (перевод из шестнадцатеричной в десятичную систему) применяется описанная ранее общая формула (все числа и вычисления – в десятичной системе).

Пример перевода шестнадцатеричного числа $1234,CF_{16}$ в десятичную систему счисления:

$$\begin{aligned} 1234,CF_{16} &= \\ 1 \times 16^3 + 2 \times 16^2 + 3 \times 16^1 + 4 \times 16^0 + 12 \times 16^{-1} + 15 \times 16^{-2} &= \\ 1 \times 4096 + 2 \times 256 + 3 \times 16 + 4 \times 1 + 12/16 + 15/256 &= \\ 4096 + 512 + 48 + 4 + 0,3125 + 0,01953125 &= 4660,80859375_{10}. \end{aligned}$$

Для обратного перевода из десятичной в шестнадцатеричную систему надо представить исходное десятичное число в виде суммы произведений базисных чисел шестнадцатеричной системы счисления на целочисленные степени числа 16 и затем записать эти числа в соответствующих позициях.

Например:

$$58,125_{10} = 3 \times 16^1 + 10 \times 16^0 + 2 \times 16^{-1},$$

следовательно, в первой позиции восьмеричного числа будет записана цифра 3, в нулевой позиции – цифра А, а в «минус первой» позиции – цифра 2. В результате получим: $58,125_{10} = 3A,2_{16}$.

Для перевода из двоичной системы счисления в шестнадцатеричную можно использовать уже знакомую формулу, однако при этом следует иметь в виду, что нумеровать позиции и выполнять все вычисления следует в шестнадцатеричной системе счисления:

$$\begin{aligned} 100110011,101_2 &= 2^8 + 2^5 + 2^4 + 2^1 + 2^0 + 2^{-1} + 2^{-3} = \\ 100_{16} + 20_{16} + 10_{16} + 2_{16} + 1_{16} + 8/10_{16} + 2/10_{16} &= 133,A_{16}. \end{aligned}$$

Таблица 3.2 иллюстрирует представление чисел в позиционных системах счисления: десятичной, двоичной, восьмеричной и шестнадцатеричной.

Анализ числовых данных, приведенных в таблице 3.2, позволяет сделать ряд полезных замечаний:

1. Чем больше основание системы счисления, тем компактнее в ней запись числа.
2. Десятичные числа, представляющие целые степени двойки, записываются в двоичной, восьмеричной и шестнадцатеричной системах круглыми числами, что является следствием того, что основания восьмеричной и шестнадцатеричной систем счисления сами являются целыми степенями двойки.

Таблица 3.2 – Представление чисел в позиционных системах счисления³

10-тичная	2-ичная	8-ричная	16-ричная
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	15	E
15	1111	17	F
$2^1 = 2$	10	2	2
$2^2 = 4$	100	4	4
$2^3 = 8$	1000	10	8
$2^4 = 16$	10000	20	10
$2^5 = 32$	100000	40	20
$2^6 = 64$	1000000	100	40
$2^7 = 128$	10000000	200	80
$2^8 = 256$	100000000	400	100
$2^9 = 512$	1000000000	1 000	200
$2^{10} = 1 024$	10000000000	2 000	400
$2^{11} = 2 048$	100000000000	4 000	800
$2^{12} = 4 096$	1000000000000	10 000	1 000
$2^{20} = 1 048 576$		4 000 000	100 000
$2^{30} = 1 073 741 824$		10 000 000 000	40 000 000

³ Содержимое этой таблицы (как минимум, ее первые 17 строк) придется выучить наизусть и помнить так же (хорошо), как школьную таблицу умножения.

3. Двоичная запись десятичных чисел, представляющих целые степени двойки, всегда содержит единицу в старшем бите и нули во всех остальных битах, количество которых равно степени двойки соответствующего десятичного числа.

4. Для двоичной записи любого одноразрядного (базисного) восьмеричного числа потребуется не более 3-х битов ($8 = 2^3$).

5. Для двоичной записи любого одноразрядного (базисного) десятичного или шестнадцатеричного числа потребуется не более 4-х битов ($2^3 < 10 < 2^4$, $16 = 2^4$).

3.2.2.5 Смешанные системы счисления

Смешанная система, по существу, не является системой счисления – скорее, это просто удобный способ записи чисел одной системы счисления с использованием набора цифр другой системы.

Смешанными (P - Q -ми) называют системы, в которых для записи чисел, заданных в одной позиционной системе счисления (с основанием, равным Q), используют набор цифр другой позиционной системы счисления (с основанием, равным P). При этом предполагается, что $P < Q$, и каждое базисное число, представленное Q -ичной системе счисления, записывается фиксированным количеством разрядов числа в P -ичной системе с сохранением порядка следования разрядов.

Пример 1: представление десятичных чисел в двоичной и двоично-десятичной системах.

Например, десятичное число $307,625_{10}$ будет представлено в двоично-десятичной ($P = 2$, $Q = 10$) системе следующим образом:

$$307,625_{10} = 0011\ 0000\ 0111, 0110\ 0010\ 0101_{2-10}.$$

Как видим, для записи каждого базисного десятичного числа используется соответствующее ему четырехразрядное двоичное число.

Процедура восстановления десятичного числа по его двоично-десятичной записи очень проста: необходимо последовательно (справа налево для целой части числа и слева направо – для дробной) каждую четверку битов преобразовать в соответствующее одноразрядное десятичное число.

Заметим, что двоичный эквивалент исходного десятичного числа не совпадает с его двоично-десятичной записью:

$$307,625_{10} = 100110011,101_2 \neq 11\ 0000\ 0111,0110\ 0010\ 0101_{2-10}.$$

Рассмотрим еще несколько примеров с двоично-восьмеричными и двоично-шестнадцатеричными числами.

Пример 2: представление восьмеричных чисел в двоичной и двоично-восьмеричной системах. Переведем двоичное число $100110011,101_2$ в восьмеричную систему счисления:

$$\begin{aligned}100110011,101_2 &= \\2^{10} + 2^5 + 2^4 + 2^1 + 2^0 + 2^{-1} + 2^{-3} &= \\400_8 + 40_8 + 20_8 + 2_8 + 1_8 + 4/10_8 + 1/10_8 &= 463,5_8\end{aligned}$$

Запишем полученное восьмеричное число $463,5_8$ в двоично-восьмеричной системе счисления – каждый разряд исходного числа будем записывать соответствующим трехразрядным двоичным числом:

$$463,5_8 = 100\ 110\ 011,101_{2-8}.$$

Заметим, что двоичная и двоично-восьмеричная записи восьмеричного числа $463,5_8$ оказались одинаковыми (в отличие от рассмотренного выше примера с десятичным числом).

Пример 3: представление шестнадцатеричных чисел в двоичной и двоично-шестнадцатеричной системах.

Представим двоичное число $100110011,101_2$ в шестнадцатеричной системе счисления: $100110011,101_2 =$

$$\begin{aligned}2^8 + 2^5 + 2^4 + 2^1 + 2^0 + 2^{-1} + 2^{-3} &= \\100_{16} + 20_{16} + 10_{16} + 2_{16} + 1_{16} + 8/10_{16} + 2/10_{16} &= 133, A_{16}.\end{aligned}$$

Запишем полученное шестнадцатеричное число $133, A_{16}$ в двоично-шестнадцатеричной системе счисления – каждый разряд исходного числа будем записывать соответствующим четырехразрядным двоичным числом:

$$133, A_{16} = 0001\ 0011\ 0011,1010_{2-16}$$

Как видим, двоично-шестнадцатеричная и двоичная записи этого шестнадцатеричного числа совпали, так же, как это было и в случае с восьмеричной системой, и это не простое совпадение, а следствие определенного соотношения оснований рассмотренных систем счисления (2, 8 и 16). Примем (без доказательства) следующее утверждение:

Если в паре позиционных систем счисления основание одной из них (Q) является целочисленной степенью основания другой (P), то Q-ичная запись любого P-ичного числа будет совпадать с его записью в смешанной (P-Q)-ичной системе.

В случае с двоично-десятичной записью число 10 не является целой степенью числа 2, а для двоично-восьмеричной и двоично-шестнадцатеричной систем указанное соотношение выполняется, следовательно, можно смело применять простую процедуру записи двоичных чисел в смешанных двоично-восьмеричной и двоично-шестнадцатеричной системах счисления для перевода восьмеричных и шестнадцатеричных чисел в двоичную систему и обратного перевода двоичных чисел в восьмеричную и шестнадцатеричную системы.

Учитывая простоту такого перевода, а также тот факт, что по сравнению с двоичной, шестнадцатеричная запись чисел в 4 раза более компактна, становится понятной популярность шестнадцатеричной системы счисления среди компьютерных специалистов (а также среди разработчиков инструментальных средств, предназначенных для компьютерных специалистов) для ее использования в качестве удобного средства краткой записи двоичных чисел.

Дело в том, что любая информация, в какой бы форме она не предъявлялась пользователю различными программными приложениями, в памяти компьютера представлена в числовом двоичном коде и обрабатывается вычислительными устройствами в двоичной системе счисления. Однако, двоичный код крайне неэффективен для визуализации компьютерных данных, их анализа и обработки человеком, в том числе и по причине большой длины такой записи.

Шестнадцатеричная запись для представления двоичных чисел широко применяется в технической документации, научных публикациях и учебной литературе по компьютерной тематике, многие инструментальные программные средства (в том числе и программы-анализаторы памяти компьютера, рекомендованные к применению при выполнении лабораторных работ) используют шестнадцатеричный формат представления двоичных чисел.

3.3 Двоичное кодирование текстовой информации

Необходимость стандартизации в области обмена текстовыми документами очевидна и явно осознавалась человечеством с древнейших времен: любой письменный язык с его алфавитом, набором синтаксических правил и правил орфографии – это своеобразный стандарт, обеспечивающий возможность информационного обмена между людьми, владеющими этим языком. С появлением технических средств приема-передачи текстов (например, телеграфа), а позднее – и компьютерных систем обработки и хранения информации, интенсивность информационных обменов возросла тысячекратно, и проблема стандартизации в этой области получила международный характер.

Наличие стандарта числового кодирования текстовых символов позволяет хранить тексты в памяти компьютеров, обрабатывать (редактировать) и визуализировать их программными средствами, то есть преобразовывать коды символов в их графические образы. Один из способов визуализации (*знакогенерации*) символов будет рассмотрен в следующей главе учебного пособия при обсуждении алгоритмов работы видеосистемы ПК в текстовых видеорежимах.

3.3.1 Стандарт ASCII

Стандарт *ASCII* (American Standard Code for Information Interchange) широко используется разработчиками компьютерной техники, системного и прикладного программного обеспечения.

Стандарт базируется на 8-битной системе кодирования символов некоторого алфавита (1 символ – 1 байт), что позволяет одновременно поддерживать алфавит мощностью не более 256 символов ($2^8=256_{10}$ или 100_h), символы которого могут иметь коды в диапазоне от 00 до FF_h. Такого количества символов достаточно для представления алфавитов любых двух европейских языков (строчные и прописные буквы), арабских цифр, знаков пунктуации, знаков математических операций и некоторых специальных символов.

Стандарт предлагает набор из 256 символов (таблица 3.3), каждому из которых поставлен в соответствие двоичный код. Все кодовое пространство разделено на два равных диапазона, называемых *кодowymi таблицами*: *основная кодовая таблица* содержит 128 символов с кодами в диапазоне от 00 до 7F; *дополнительная кодовая таблица* содержит 128 символов с кодами в диапазоне от 80_h до FF_h. При этом предполагается, что основная таблица присутствует всегда, а дополнительных таблиц может быть много, все они спроецированы на единое кодовое пространство, но лишь одна из них может быть активной.

Основная кодовая таблица содержит коды букв латинского алфавита, цифр, знаков пунктуации знаков математических операций, а также некоторых специальных символов.

Основное назначение дополнительной кодовой таблицы – кодирование символов какого-либо (одного !) национального алфавита и так называемых символов псевдографики – специальных знаков, предназначенных для отображения границ таблиц (здесь надо не забывать о том, стандарт создавался во времена матричных принтеров и текстовых режимов работы видеоадаптеров и экранов компьютерных мониторов).

Таблица 3.3 – Кодовые таблицы стандарта 8-битного кодирования символов

Основная кодовая таблица ASCII		Дополнительная кодовая таблица (кириллица в так называемой «альтернативной» кодировке)	
Диапазон кодов	Набор символов	Диапазон кодов	Набор символов
00 – 1F	Управляющие символы : <i>нуль-символ</i> , ☺, ☹, ♥, ♦, ♣, ♠, •, ◻, ◯, ◼, ♂, ♀, ♪, ♫, ☀, ▶, ◀, ↕, ↗, !!, !!, ¶, §, —, ↕, ↑, ↓, →, ←, L, ↔	80 – 9F	Прописные буквы (кириллица): А, Б, В, Г, Д, Е, Ж, ... Я (кроме Ё)
20 – 2F	Знаки пунктуации: <i>пробел</i> , !, ", #, \$, %, &, ', (,), *, +, ,, -, ., /	A0 – AF	Строчные буквы (кириллица от а до п): а, б, в, г, д, е, ж, ... п (кроме буквы ё)
30 – 39	Арабские цифры: 0, 1, 2, 3, ..., 9	B0 – DF	Символы псевдографики: ⌠, †, ‡, ¶ и др.
3A – 40	Знаки: :, ;, <, =, >, ?, @	E0 - EF	Строчные буквы (кириллица от р до я): р, с, т, у, ф, х, ц, ...я
41 – 5A	Прописные буквы (латиница): A, B, C, D, ..., Z	F0 – F1	Буквы Ё и ё (кириллица)
5B – 60	Знаки: [, \,], ^, _ , `	F2 – F7	Буквы славянских алфавитов, отсутствующие в русском алфавите : Є, е, Ї, і, Ў, ў
61 – 7A	Строчные буквы (латиница): a, b, c, d, ..., z	F8 – FF	Знаки: °, ·, ;, √, №, α, ■, ■,
7B – 7F	Знаки: {, , }, ~, △		

Информацию о содержании кодовых таблиц можно легко получить, используя любой текстовый редактор: для этого надо последовательно вводить коды символов (в десятичной системе счисления), нажимая на клавиши «цифровой клавиатуры» и удерживая нажатой клавишу Alt – после отпускания клавиши Alt на экран будет выведен символ, соответствующий набранному коду.

Американский национальный стандарт ASCII, ставший де-факто международным (по вполне понятным причинам), был далеко не единственным стандартом кодирования текстовых данных – одновременно с ним использовались и многие другие стандарты 8-битного кодирования символов (например, советский стандарт КОИ-8).

Все эти стандарты структурно были одинаковы, отличаясь только соответствием символов алфавитов их двоичным кодам, и при этом все они имели общий недостаток – этих кодов (и, соответственно – символов) было всего 256 (2^8), из которых половина всегда была «занята» символами латиницы, цифрами и другими «международными» знаками, и только 128 кодов оставалось для символов какого-то другого алфавита.

Несомненным достоинством всех этих 8-битных стандартов была высокая экономичность хранения текстовых данных, что было совсем немаловажно в эпоху, «когда компьютеры были большими, а запоминающие устройства – маленькими», но со временем проявились два существенных недостатка 8-битного кодирования символов.

Во-первых, компьютер оказывался только «двуязычным», и было весьма проблематично хранить и обрабатывать на одном компьютере текстовые документы, написанные более, чем на двух языках (точнее – более, чем на одном неанглийском языке). Еще большие проблемы возникали в ситуации, когда в один документ требовалось включать разно-языковые фрагменты.

Второй недостаток связан с ограничением мощности используемого алфавита. Носителям языков европейской группы (как, впрочем, и некоторых других языков) сильно повезло – в алфавитах этих языков всего порядка 30 знаков, с учетом регистров – порядка 60, что хорошо укладывается в предельно-допустимое значение 128 символов для дополнительной кодовой таблицы.

Совсем другая ситуация создается при кодировании иероглифических алфавитов. Так, например, разные словари китайских иероглифов содержат *от 50 до 85 тысяч* знаков, в большом японском словаре приведено *50 тысяч* иероглифов, из которых в современном языке их чуть больше *трех тысяч*, и только (!) *1945* иероглифов утверждены в качестве повседневно-необходимых.

Разумеется, можно ограничиться техническим китайским или японским алфавитами и успешно вести на таких усеченных языках деловой документооборот, писать письма друзьям, газетные репортажи и научные статьи, можно даже написать руководство по эксплуатации велосипеда или персонального компьютера, но вряд удастся записать текст поэтического произведения без использования всего богатства этих великих восточных языков.

Естественным выходом из такой ситуации является отказ от 8-битной системы кодирования, что и было сделано разработчиками стандарта *UNICODE*.

3.3.2 Стандарт UNICODE

Стандарт Unicode был разработан в 1991 году с целью создания единой кодировки символов всех современных и многих древних письменных языков. Каждый символ в этом стандарте представляется 16-разрядным двоичным числом, что позволяет закодировать 65536 символов (что явно должно удовлетворить высокообразованных носителей китайского и японского языков).

В одном текстовом документе, закодированном в стандарте *Unicode*, могут одновременно присутствовать китайские иероглифы, математические символы, буквы греческого алфавита, латиницы и кириллицы, при этом становится ненужным переключение кодовых страниц.

Другим существенным отличием стандарта Unicode является то, что он не только приписывает каждому символу уникальный код, но и определяет различные характеристики этого символа, например:

- тип символа (прописная или строчная буква, цифра, знак препинания);
- атрибуты символа (отображение слева направо или справа налево, разрыв строки и т.д.);
- соответствующая прописная или строчная буква (для строчных и прописных букв);
- соответствующее числовое значение (для цифровых символов).

Стандарт состоит из двух основных разделов: универсальный набор символов *UCS (Universal Character Set)*, определяющий однозначное соответствие символов их двоичным кодам, и семейство кодировок *UTF (Unicode Transformation Format)*, определяющее машинное представление последовательности кодов *UCS*.

Всё кодовое пространство в диапазоне кодов от 0000 до FFFF разбито на несколько *стандартных подмножеств*, каждое из которых соответствует либо алфавиту какого-то языка, либо группе специальных символов, сходных по своим функциям:

- диапазон с кодами от 0000 до 00FF содержит символы основной и дополнительной кодовых таблиц ASCII;
- далее расположены знаки различных письменностей, знаки пунктуации и технические символы;
- под символы кириллицы выделен диапазон с кодами от 0400 до 04FF.
- часть кодов зарезервирована для использования в будущем.

В таблице 3.4 приведен (исключительно для демонстрации возможностей стандарта) перечень подмножеств Unicode 3.0. с указанием диапазонов кодов.

Таблица 3.4 – Подмножества стандарта UNICODE

Диапазон кодов	Описание подмножества	Диапазон кодов	Описание подмножества
00 – 7F	Базовая латиница	2150 – 218F	Числовые символы
80 – FF	Латиница-1 дополнительная	2190 – 21FF	Стрелки
100 – 17F	Расширенная латиница - A	2200 – 22FF	Математические операторы
180 – 24F	Расширенная латиница - B	2300 – 23FF	Технические символы
250 – 2AF	Фонетические знаки	2400 – 243F	Значки управляющих кодов
2B0 – 2FF	Символы изменения пробела	2440 – 245F	Оптическое распознавание образов
300 – 36F	Диакритические знаки	2460 – 24FF	Буквы и цифры в кружочках
370 – 3FF	Греческое и коптское письмо	2500 – 257F	Символы рамок
400 – 4FF	Кириллица	2580 – 259F	Символы заполнения
530 – 58F	Армянское письмо	25A0 – 25FF	Геометрические символы
590 – 5FF	Ивритское письмо	2600 – 26FF	Различные значки
600 – 6FF	Арабское письмо	2700 – 27BF	Значки Dingbats
700 – 74F	Сирийское письмо	2800 – 28FF	Шрифт Брайля
780 – 7BF	Тана (мальдивское письмо)	2E80 – 2EFF	Доп. ключи к иероглифам
900 – 97F	Деванагари	2F00 – 2FDF	Кандзи (ключи к иероглифам)
980 – 9FF	Бенгали	2FF0 – 2FFF	Описания идеограмм
A00 – A7F	Гурмукхи	3000 – 303F	Идеографические символы и знаки препинания
A80 – AFF	Гуджарати	3040 – 309F	Хирагана (японское письмо)
B00 – B7F	Ория	30A0 – 30FF	Катакана (японское письмо)
B80 – BFF	Тамили	3100 – 312F	Китайское слоговое письмо
C00 – C7F	Телугу	3130 – 318F	Корейское совместимое письмо
C80 – CFF	Каннада	3190 – 319F	Камбун
D00 – D7F	Малаялам	31A0 – 31BF	Китайское расширенное слоговое письмо

Окончание таблицы 3.4

Диапазон кодов	Описание подмножества	Диапазон кодов	Описание подмножества
D80 – DFF	Сингальское письмо	3200 – 32FF	Дальневосточные буквы и месяцы в кружочках
E00 – E7F	Тайское (сиамское) письмо	3300 – 33FF	Дальневосточные совместимые символы
E80 – EFF	Лаосское письмо	3400 – 4DB5	Унифицированные иероглифы, расширение А
F00 – FFF	Тибетское письмо	4E00 – 9FFF	Унифицированные иероглифы
1000 – 109F	Бирманское письмо	A000 – A48F	Слоговое письмо ицзу (и)
10A0 – 10FF	Грузинское письмо	A490 – A4CF	Ключи письма ицзу (и)
1100 – 11FF	Корейское письмо	AC00 – D7A3	Корейское слоговое письмо
1200 – 137F	Эфиопское письмо	D800 – DB7F	Первый байт суррогатных символов
13A0 – 13FF	Чероки письмо	DB80 – DBFF	Первый байт личных суррог. симв.
1400 – 167F	Письмо алгонкинских народов	DC00 – DFFF	Второй байт суррогатных символов
1680 – 169F	Огамическое письмо	E000 – F8FF	Область личных символов
16A0 – 16FF	Руническое письмо	F900 – FAFF	Совместимые иероглифы
1780 – 17FF	Кхмерское письмо	FB00 – FB4F	Декоративные варианты букв
1800 – 18AF	Монгольское письмо	FB50 – FDFE	Арабское декоративное письмо - А
1E00 – 1EFF	Дополнительная латиница	FE20 – FE2F	Диакритические знаки половинной ширины
1F00 – 1FFF	Греческое расширенное письмо	FE30 – FE4F	Декоративные дальневосточные формы
2000 – 206F	Знаки пунктуации	FE50 – FE6F	Малые варианты символов
2070 – 209F	Верхние и нижние индексы	FE70 – FEFE	Арабское декоративное письмо - В
20A0 – 20CF	Денежные символы	FEFF	Неразрывный пробел нулевой ширины
20D0 – 20FF	Диакритические знаки символов	FF00 – FFEF	Символы полной и половинной ширины
2100 – 214F	Буквенные символы	FFF0 – FFFD	Специальные символы

Обсуждая стандарты двоичного кодирования текстовых символов, мы оцениваем качество и возможности этих стандартов по мощности поддерживаемых ими алфавитов, затратам памяти на хранение закодированных текстов и при этом обращаем внимание на удобство интерпретации кодов символов в процессе их визуализации.

При всем разнообразии таких стандартов, алгоритм интерпретации будет примерно одинаков и весьма прост:

- транслятор, получив для обработки код очередного символа, последовательно просматривает бинарную кодовую таблицу (похожую на таблицу 3.3 или 3.4);
- находит в левом столбце одной из ее строк искомый код, и «видит» в правом столбце этой строки соответствующий этому коду символ;
- «рисует» этот символ, используя, например, соответствующую пиксельную матрицу его графического образа.

3.4 Двоичное кодирование десятичных чисел

Основным критерием оценки качества кодов числовых данных является эффективность выполнения вычислительных операций с этими кодами. Не вдаваясь в детали аппаратной реализации арифметических операций, заметим, что процессор компьютера выполняет элементарные арифметические операции совсем не так, как это делает человек, и эффективность машинной реализации операций во многом зависит от формы представления операндов – числовых данных, хранимых в памяти компьютера в определенных форматах.

В контексте рассматриваемого вопроса *числовыми данными* будем называть операнды и результаты вычисления математических выражений – константы базовых (скалярных) числовых типов или переменные, получившие числовые значения таких типов.

Если переменная получила числовое значение (например, в результате выполнения операции присваивания), то это значение будет сохранено в определенной «ячейке» памяти, формат которой соответствует *типу данных* этой переменной.

Форматом хранения значения переменной числового типа будем называть двоичный код, соответствующий фактическому значению модуля кодируемого числа, его знаку (для «знаковых» типов данных) и/или точности представления (для вещественных чисел), а также диапазону допустимых значений переменной.

3.4.1 Кодирование натуральных чисел

Если для переменной определен тип данных «целое без знака» (например, тип `unsigned int` в языке *C* или типы `Byte / Word` в языке *Pascal*), значит, программист уверен в том, что при всем многообразии значений этой переменной (на то она и *переменная*), любое из них будет, во-первых, целым числом и, во-вторых, это число не будет отрицательным.

Определение 1: Машинным представлением натурального десятичного числа является его **прямой двоичный код**, получаемый в результате перевода этого числа в двоичную систему счисления по правилам арифметики.

Для определения формата хранения натурального десятичного числа N потребуется последовательно выполнить ряд простых операций:

1). *Определить диапазоном допустимых значений* этой переменной в соответствии с содержательной постановкой решаемой задачи (точнее – определить только ее наибольшее возможное значение N_{\max}).

2). *Вычислить разрядность* (в битах) прямого двоичного кода, достаточную для хранения максимального значения N_{\max} : $L_{\text{bit}} = \text{Log}_2 N_{\max}$, округляя результат вычисления до ближайшего большего целого.

3). *Определить размер ячейки памяти* (в байтах): $L_{\text{byte}} = L_{\text{bit}} / 8$, округляя результат вычисления до ближайшего большего целого.

4). Выбрать соответствующий *тип данных*, поддерживаемый используемым языком программирования, например, тип `UnSignedINT`, `Byte` или `Word`. Заметим, что использование языков с динамической типизацией (например, *Visual Basic*, *Python* или *Erlang*) избавляет программиста от необходимости «типизации» переменных – соответствующие типы данных автоматически («динамически») определяются (не для переменных, а для их числовых значений) в момент выполнения операции присваивания.

Если для хранения натурального числа выделена ячейка памяти размером в m байтов, то максимальное значение этого числа $N_{\max} = 2^{8 \cdot m} - 1$ (таблица 3.5).

Таблица 3.5 – Примеры кодирования натуральных десятичных чисел

Десятичное число	Однobaйтовый формат		Двухбайтовый формат	
	Binary	Hexadecimal	Binary	Hexadecimal
0	0000 0000	00	0000 0000 0000 0000	0000
128	1000 0000	80	0000 0000 1000 0000	0080
255	1111 1111	FF	0000 0000 1111 1111	00FF
65535	-	-	1111 1111 1111 1111	FFFF

3.4.2 Кодирование целых чисел со знаком

Для целочисленных переменных, которые могут получать при выполнении программы как положительные, так и отрицательные значения, предусмотрены специальные форматы двоичного представления, поддерживаемые соответствующими числовыми типами данных, например, *Integer* или *Long*.

В таких форматах обеспечиваются эффективные способы хранения информации как о модуле, так и о знаке кодируемого десятичного числа, при этом под эффективностью здесь понимается «удобство» машинной реализации арифметических операций, человеку же такие форматы покажутся крайне неудобными и непривычными.

Модели вычислений, удобные человеку, хорошо знакомы всем нам с начальной школы. Очевидно, самой простой является *аналоговая модель*, в основе которой понятие *числовой оси* с линейной шкалой. В этой модели аналогом числа является отрезок числовой оси: модуль числа представлен длиной отрезка, а знак числа – его направлением: положительные числа ассоциируются с отрезками, направленными вправо, а отрицательные – с отрезками, направленными влево. На такой модели алгоритм вычисления алгебраической суммы двух слагаемых реализуется «геометрически»:

а) отрезок, представляющий первое слагаемое, «откладывается» от нулевой точки оси в направлении, соответствующей его знаку;

б) от конца первого отрезка откладывается отрезок второго слагаемого в соответствии с его знаком;

в) результат вычисления суммы будет представлен отрезком, направленным от нулевой точки числовой оси к концу второго отрезка: если результирующий отрезок направлен вправо, то сумма положительна, в противном случае – отрицательна, при этом длина результирующего отрезка представляет модуль полученной суммы.

Ограничения такой аналоговой модели очевидны⁴, гораздо более эффективна для «ручного» счета дискретная (цифровая) модель поразрядных вычисле-

⁴ При всей ограниченности этой модели, на ее основе была создана самая массовая и некогда популярная в инженерной среде аналоговая вычислительная машина – *логарифмическая линейка*, в которой использовались не линейные, а логарифмические шкалы числовых осей, что позволило реализовать операции умножения / деления чисел через операции сложения / вычитания их логарифмов ($\text{Log}[a \times b] = \text{Log}[a] + \text{Log}[b]$ и $\text{Log}[a/b] = \text{Log}[a] - \text{Log}[b]$).

ний, использующая такие базовые свойства позиционных k -ичных систем счисления, как «позиционные веса» базисных чисел и объединение k единиц младшего разряда в одну единицу следующего старшего разряда.

Цифровая модель положена в основу «ручных» алгоритмов поразрядных вычислений, известных как «сложение / вычитание в столбик». Машинная реализация «ручной» операции сложения не создает особых проблем: циклически, начиная с младшего разряда, складываем соответствующие базисные числа слагаемых, анализируем разрядную сумму и, если она превышает основание системы счисления, вычитаем из суммы основание, переносим единицу в следующий старший разряд и используем ее в качестве третьего слагаемого на следующем проходе по циклу.

Операция поразрядного вычитания реализуется аналогично (с использованием «заёмных» единиц старших разрядов), но только в том случае, когда уменьшаемое больше вычитаемого, и конечный результат операции оказывается положительным. В противном случае (когда большее число вычитается из меньшего) алгоритм существенно усложняется: уменьшаемое и вычитаемое меняются местами, выполняется противоположная заданной операция вычитания, и к полученному (положительному) результату слева добавляется специальный знак «минус», показывающий, что разность операндов – отрицательна.

Заметим, что машинная реализация такого алгоритма вычитания привела бы к двукратной потере производительности вычислений, так как для определения большего из двух n -разрядных операндов потребуется выполнить n операций поразрядного сравнения, суммарная стоимость которых соизмерима со стоимостью основной операции поразрядного вычитания.

И остается еще одна проблема, требующая решения при разработке компьютерного формата кодирования «чисел со знаком» – это проблема знака «минус» перед обозначением модуля отрицательного числа.

Стандартный формат, используемый для компьютерного представления целых чисел со знаком, предусматривает хранение модулей отрицательных чисел в *дополнительном двоичном коде*, что исключает необходимость выполнения машинной операции вычитания, заменяя ее операцией сложения.

Определение 2: *Дополнительный код n -разрядного числа, заданного в любой позиционной системе счисления – это такое n -разрядное число, которое дополняет кодируемое число до $(n+1)$ -разрядного числа, содержащего единицу в старшем разряде, и нули – во всех остальных разрядах.*

Примеры формирования дополнительных кодов n -разрядных чисел, заданных в десятичной и двоичной системах счисления, приведены в таблице 3.6.

Таблица 3.6 – Примеры формирования дополнительных кодов чисел

Десятичная система счисления			Двоичная система счисления		
Число в прямом коде	Дополнительный код числа		Число в прямом коде	Дополнительный код числа	
	при $n = 3$	при $n = 5$		при $n = 7$	при $n = 15$
1	999	99999	1	1111111	111111111111111
100	900	99900	1010	1110110	111111111110110
500	500	99500	1010101	0101011	111111110101011
999	1	99001	1111111	0000001	111111110000001
50000	-	50000	111111111	-	111110000000001

Как следует из *определения 2* и примеров из таблицы 3.6, сумма прямого и дополнительного кодов n -разрядного числа равна $(n+1)$ -разрядному числу, содержащему единицу в старшем разряде и нули – во всех остальных. Из этого следует простой способ формирования дополнительного кода числа по его прямому коду путем вычитания n -разрядного прямого кода числа из $(n+1)$ -разрядного «круглого» числа.

Такой способ применим для любой позиционной системы счисления, однако, для двоичной системы существует и другой способ, более удобный для устного счета и реализуемый двумя последовательными шагами:

- 1) n -разрядный *прямой код* преобразуется в *обратный код* (называемый также *инверсным*) путем замены в прямом коде всех нулей на единицы, а единиц – на нули;
- 2) *дополнительный код* формируется из полученного *обратного кода* путем прибавления к нему числа «1» по правилам арифметики.

Приведем правила компьютерного кодирования целых чисел со знаком:

- 1). Для хранения информации о знаке числа используется старший бит из общего количества битов, выделенных для хранения двоичного кода числа. Этот бит называется «знаковым» и содержит «1» для отрицательных чисел и «0» – для положительных чисел и числа «0».

2). Остальные (младшие) биты используются для хранения двоичного кода модуля числа. Например, если для хранения целого числа со знаком выделена ячейка памяти размером в 1 байт, то для кодирования кода модуля этого числа можно использовать только 7 младших битов.

3). Если кодируемое число – положительно, то модуль числа кодируется **прямым двоичным кодом** подобно тому, как это делается при кодировании натуральных чисел. Например, использование ячейки памяти размером в 1 байт не позволит хранить в ней положительные числа, превышающие значение $+127_{10}$ (01111111_2).

4). Если кодируемое число – отрицательно, то модуль числа кодируется **дополнительным двоичным кодом**.

Рассмотрим несколько примеров, иллюстрирующих систему кодирования целых чисел со знаком.

Первый пример иллюстрирует тот факт, что использование дополнительного кода для представления отрицательного числа позволяет заменить операцию вычитания операцией сложения. Рассмотрим хорошо знакомую операцию поразрядного вычитания («в столбик») двух трехразрядных десятичных чисел:

$$782 - 234 = 548$$

Здесь уменьшаемое больше вычитаемого, порядок следования операндов в операции оставлен без изменения, и результат операции положителен.

Рассмотрим операцию вычитания, противоположную предыдущей:

$$234 - 782 = -(782 - 234) = -548.$$

Здесь уменьшаемое меньше вычитаемого, и поэтому вместо исходной операции вычитания « $234 - 782$ » выполнена противоположная ей операция « $782 - 234$ » и к полученному промежуточному результату « 548 » добавлен специальный знак «минус» перед модулем числа для обозначения того факта, что результат операции « -548 » – отрицателен.

Сделаем в связи с эти несколько полезных замечаний:

- а) число « -548 » с определенной долей условности обозначает отрицательный результат выполнения исходной операции вычитания « $234 - 782$ »;
- б) в записи отрицательного результата мы использовали специальный символ – знак «минус» перед модулем числа;

в) для получения этого результата нам пришлось предварительно выполнить операцию поразрядного сравнения операндов, стоимость которой соизмерима со стоимостью основной операции, и по результатам сравнения принять решение о замене порядка следования операндов;

г) реализация вычислительным устройством такого алгоритма выполнения операции вычитания (перед каждой операцией – поразрядное сравнение операндов) привела бы к двукратной потере производительности вычислений.

Попытаемся выполнить исходную операцию вычитания «234 – 782» без предварительного сравнения и замены порядка следования операндов по алгоритму поразрядного вычитания «в столбик», начиная с младших разрядов, занимая, при необходимости, единицу в старшем разряде и превращая ее в десяток в младшем:

$$4 - 2 = 2$$

$$[10] + 3 - 8 = 5$$

$$[10] + 1 - 7 = 4$$

Заметим, что все действия выполнения в строгом соответствии с принципом позиционности десятичной системы счисления. Получен результат *452, где «452» – числовой код, представляющий модуль результата операции, а знак «*» – специальный символ, обозначающий тот факт, что результат операции отрицателен, так как при вычитании старших разрядов нам пришлось занять единицу в несуществующем ($n+1$)-м разряде уменьшаемого.

Сравним результаты выполнения этой операции: числа «-548» и «*452» – это две разных формы условной записи одного и того же числа. Заметим, что число 452 *дополняет* число 548 до числа 1000, то есть в сумме эти два числа дают 1000. Другими словами, число 452 – это дополнительный код числа 548.

Продолжим арифметический эксперимент – выполним операцию алгебраического сложения некоторого числа с результатом предыдущей операции:

- 1) вычитание прямого кода числа: $800 - 548 = 252$;
- 2) сложение с дополнительным кодом числа: $800 + 452 = (1)252$;
- 3) вычитание прямого кода числа: $300 - 548 = -248$;
- 4) сложение с дополнительным кодом числа: $300 + 452 = 752$

В 1-й и 2-й операциях получен одинаковый результат – трехразрядное положительное число 252. При сложении с дополнительным кодом в старшем (не существующем) разряде числа получена единица – это разряд переполнения и будет отброшен, однако его наличие может использоваться в качестве признака положительного результата и, как следствие – признаком того, что полученный результат представлен в прямом коде.

В результате выполнения 4-й операции разряд переполнения результата отсутствует, что может служить признаком отрицательного результата, и, как следствие, признаком того, что число 752 представляет полученный результат в дополнительном коде, соответствующем результату «-248» 3-й операции.

Следующие примеры иллюстрируют реализацию машинных операций вычитания в двоичном коде (для краткости используется однобайтовый формат хранения данных).

Второй пример: вычитание с положительным результатом.

Операция вычитания десятичных чисел $7_{10} - 1_{10} = 6_{10}$ в ее машинном представлении будет заменена операцией сложения двоичных кодов модулей операндов $7_{10} + (-1_{10})$, в которой модуль положительного операнда будет представлен в прямом коде, а модуль отрицательного – в дополнительном (старшие знаковые биты двоичных кодов подчеркнуты):

- машинное представление числа «+7₁₀»: 00000111₂ (нулевой знаковый бит и прямой семиразрядный двоичный код модуля этого числа);
- прямой 7-разрядный двоичный код модуля числа 1₁₀: 000 000₂;
- обратный 7-разрядный двоичный код модуля числа 1₁₀: 111 111₂;
- дополнительный двоичный код модуля числа 1₁₀: 111 111₂;
- машинное представление числа «-1₁₀» (единичный знаковый бит и дополнительный 7-разрядный код модуля): 1111 111₂.

Операция *вычитания* ($7 - 1$) заменена операцией *сложения* машинных кодов ее операндов: положительное слагаемое «+7», модуль которого представлен прямым кодом, и отрицательное слагаемое «-1», модуль которого представлен дополнительным кодом: 00000111₂ + 11111111₂ = 00000110₂.

Как видим, в знаковом бите результата записан 0, то есть ответ положителен, и, следовательно, 7-разрядное число 0000110₂ представляет модуль результата в прямом коде, то есть этот результат – десятичное число «+6».

Третий пример: вычитание с отрицательным результатом.

- операция в десятичной системе счисления: $1_{10} - 7_{10} = -6_{10}$.

- машинное представление числа $+1_{10}$: $\underline{0000} 0001_2$;
- прямой 7-разрядный двоичный код числа 7_{10} : $000 0111_2$;
- обратный 7-разрядный двоичный код числа 7_{10} : $111 1000_2$;
- дополнительный 7-разрядный двоичный код числа 7_{10} : $111 1001_2$;
- машинное 8-битовое представление числа « -7_{10} »: $\underline{1111} 1001_2$

(дополнительный код модуля числа 7_{10} с единицей в знаковом бите).

Операция *вычитания* ($1 - 7$) заменяется операцией *сложения* машинных кодов чисел $+1$ и -7 : $\underline{0000} 0001_2 + \underline{1111} 1001_2 = \underline{1111} 1010_2$. Как видим, результат выполнения такой операции содержит единицу в знаковом бите, следовательно, это дополнительный код некоторого отрицательного числа.

Для того, чтобы определить модуль этого числа, надо восстановить прямой код из 7-разрядного дополнительного кода (путем вычитания из него единицы и последующего поразрядного инвертирования, то есть замены нулей на единицы и единиц на нули):

- обратный код из дополнительного: $111 1010_2 - 000 0001_2 = 111 1001_2$;
- прямой код из обратного: $000 0110_2$.

Полученный результат – код десятичного числа 6, то есть результатом операции является число « -6 », что и требовалось доказать.

3.4.3 Кодирование вещественных чисел

В отличие от целых, вещественные числа используются для приближенных вычислений – они всегда представляют некоторое количество с определенной степенью точности. Эту точность принято обозначать количеством знаков, записываемых в дробной части числа. Например, два числа: 3,14 и 3,14159 описывают одну и ту же константу π с разной степенью точности.

Используется и другая (*экспоненциальная*) форма записи вещественных чисел: $M \times k^p$, где M называют *мантиссой* числа, p – *порядком* числа, а k – это основание системы счисления, в которой записано число.

Например, ту же константу π можно записать множеством различных способов: $0,314 \times 10^1$, $3,14 \times 10^0$, 314×10^{-2} или 314159×10^{-5} . Таким образом, вещественное число может быть однозначно задано тремя числами: *мантиссой*, *порядком* и *основанием системы счисления*.

Если вспомнить, что при компьютерном представлении информации всегда используется двоичная система счисления, для представления вещественного числа достаточно двух чисел – мантиссы и порядка этого числа. При этом количество разрядов, отводимых для мантиссы, определяет достижимую точность

представления результата вычисления, а количество разрядов, отводимых для порядка, определяет масштаб (диапазон допустимых значений) числа.

Ассоциацией IEEE (Institute of Electrical and Electronics Engineers) в 1985 году был разработан стандарт для представления в двоичном коде вещественных чисел (чисел с плавающей точкой). Стандарт получил обозначение **IEEE-754** и широко используется при аппаратной и низкоуровневой программной реализации вычислительных процедур. Стандарт определяет:

- правила машинного представления нормализованных и денормализованных положительных и отрицательных чисел с плавающей точкой;
- правила машинного представления числа «нуль» и специальных величин «бесконечность» и «не число»;
- правила (режимы) округления действительных чисел.

Стандартом IEEE754-85 определены два основных формата представления двоичных вещественных чисел, отличающиеся размером ячейки памяти, используемой для хранения мантиссы и порядка числа, и, соответственно, допустимыми диапазонами значений чисел и достижимой точностью вычислений:

- одинарная точность (single-precision, binary-32) – 32 бита;
- двойная точность (double-precision, binary-64) – 64 бита;

Основные форматы поддерживаются большинством языков программирования: например, *Visual Basic* поддерживает типы данных `single` и `double`, в языке *C* аналогичные типы получили название `float` (32 бита) и `double`, а *Python* поддерживает тип `float` половинной точности (half-precision, binary-16), дополнительно введенный в стандарт IEEE-754 при его пересмотре в 2008 году.

Стандартом IEEE-754 2008 года введены и другие базовые форматы представления вещественных двоичных и десятичных чисел (кроме уже упомянутого выше формата `binary-16`):

- двоичные числа четырехкратной точности (`binary-128`);
- двоичные числа восьмикратной точности (`binary-256`);
- десятичные числа одинарной точности (`decimal-32`);
- десятичные числа двойной точности (`decimal-64`);
- десятичные числа четырехкратной точности (`decimal-128`).

Рассмотрим представление нормализованных двоичных чисел в формате `single-precision (binary-32)`, так как остальные форматы являются, по сути, его увеличенными (или уменьшенными) копиями.

Для пояснения понятия нормализации / денормализации чисел вернемся к рассмотренному ранее примеру с экспоненциальной записью константы π и обратим внимание на то, что существует множество эквивалентных записей этого числа, например: $0,0314 \times 10^2$, $0,314 \times 10^1$, $3,14 \times 10^0$, 314×10^{-2} , 3140×10^{-3} . Отличаются эти записи формой представления *мантиссы* и, соответственно, значением *порядка* (называемого также *экспонентой*) числа.

Определение 3: Число, представленное в позиционной системе счисления, называют **нормализованным**, если его мантисса не меньше единицы и не больше самого большого базисного числа системы счисления.

Для десятичной системы счисления мантисса нормализованного числа должна находиться в диапазоне значений: $9 \geq M \geq 1$. В нашем примере нормализованной записью константы π является число $3,14 \times 10^0$.

Для двоичной системы счисления это неравенство примет вид: $1 \geq M \geq 1$, из чего следует, что в старшем разряде мантиссы нормализованного двоичного числа всегда будет записана единица. Например, мантисса двоичного числа $11,001001_2$ в нормализованной форме будет представлена числом $1,1001001_2$ (при этом, естественно, порядок числа будет равен +1).

Определение 4: Число называют **денормализованным**, если его мантисса находится в диапазоне значений: $1 > M \geq 0,1$, то есть в старшем разряде мантиссы денормализованного числа – всегда ноль, а в соседнем с ним младшем разряде – всегда не ноль.

В наших примерах: $0,314 \times 10^1$ – денормализованная запись константы π , а двоичное число $0,11001001_2 \times 2^2$ – денормализованное представление числа $11,001001_2$ (основание и порядок двоичного числа условно записаны в десятичной системе счисления).

Чтобы представить десятичное число в формате single-precision необходимо привести его к нормализованному двоичному виду и записать знак, порядок и мантиссу числа определенным образом в ячейку размером в 32 бита.

Представление знака числа.

Для кодирования знака числа используется старший (31-й) бит: 0 – для положительных чисел, 1 – для отрицательных.

Представление порядка числа.

Для хранения порядка числа используются следующие 8 битов (с 23-го по 30-й), причем порядок хранится в специальном *смещенном* формате: к значению

порядка числа прибавляется двоичный эквивалент числа 127 (0111111_2) и тем самым исключается необходимость хранения отрицательных порядков (и, соответственно, дополнительных кодов). Например, если порядок (после перевода числа в двоичную систему и его последующей нормализации) оказался равным $+111_2(+7)$, то храниться он будет, как число 134 ($7+127$), или, в двоичной системе счисления, как $111+0111111 = 10000110$.

При таком формате хранения порядка и положительные, и отрицательные его значения будут представлены положительными числами: например, порядок « -7 » получит значение «120» ($-7+127$) или « 01111000_2 » ($-111_2+0111111_2$).

Самый большой порядок числа « $+128$ » будет храниться, как число 255 (1111111_2), а самый маленький « -127 » – как число 0. Для того, чтобы вычислить фактическое значение порядка числа, надо вычесть число 127 из числа, хранимого в этих восьми битах.

Представление нормализованной мантиссы числа.

Нормализованная мантисса хранится в 23-х младших битах (с 0-го по 22-й) в прямом двоичном коде, независимо от знака числа. Учитывая, что старший разряд (целая часть) нормализованной мантиссы двоичного числа всегда содержит единицу, ее можно вообще не хранить (в целях экономии, так как каждый дополнительный бит удваивает диапазон значений двоичного числа), а «подразумевать» и «дописывать» при реализации вычислительных операций. Так и поступили разработчики стандарта: в младших 23-х битах хранится только дробная часть нормализованной мантиссы (как целое положительное число в прямом двоичном коде).

Рассмотрим пример стандартного 32-битового двоичного кодирования десятичного числа 155,625.

- 1) представим это число в двоичной системе счисления: $10011011,101_2$;
- 2) число положительно, следовательно, старший бит равен 0;
- 3) запишем полученное двоичное число в нормализованном экспоненциальном формате $1,0011011101_2 \times 2^7$ – мантисса $1,0011011101_2$, порядок 111_2 ;
- 4) дробная часть мантиссы: $0011011101\{000000000000\}$ – в фигурных скобках показаны незначащие нули, дополняющие дробную часть мантиссы *справа* до 23-х разрядов;
- 5) порядок числа (в «смещенном» формате): $7 + 127 = 134$, или, в двоичной системе счисления: $111_2 + 01111111_2 = 1000\ 0110_2$.

В результате получим следующее машинное представление исходного десятичного числа 155,625 в стандартном формате одинарной точности:

0100 0011 0001 1011 1010 0000 0000 0000₂,

или, в шестнадцатеричной системе счисления:

431BA000₁₆.

Машинное представление отрицательного числа –155,625 будет отличаться от рассмотренного ранее только значением старшего (знакового) бита:

1100 0011 0001 1011 1010 0000 0000 0000₂

или, в шестнадцатеричной системе счисления:

C31BA000₁₆.

Для восстановления исходного десятичного числа F по его 32-битной двоичной записи в соответствии со стандартом IEEE-754 можно воспользоваться следующей формулой:

$$F = (-1)^S \times 2^{(E-127)} \times (1 + M/2^{23})$$

где: S – знаковый бит (0 или 1); E – смещенный порядок;

M – дробная часть нормализованной мантиссы.

Отметим, что (E – 127) – это фактический порядок нормализованного числа, (1 + M/2²³) – нормализованная мантисса, где M/2²³ – ее дробная часть, полученная из целочисленной записи M, а единица – это та самая целая часть мантиссы, которая была «отброшена» при записи (конечно же, не из экономии, а для повышения точности представления результатов вычислений).

Проверим приведенную формулу на нашем примере +155,625:

- 1) S = 0, (-1)⁰ = 1;
- 2) E = 1000 0110₂ = 134;
- 3) 2⁽¹³⁴⁻¹²⁷⁾ = 2⁷ = 128;
- 4) M = 001101110100000000000000 = 1810432;
- 5) M/2²³ = 0,2158203125;
- 6) 1 + M/2²³ = 1,2158203125;
- 7) 1 × 128 × 1,2158203125 = 155,625.

Операции умножения нормализованных чисел выполняются процессором компьютера по простому алгоритму:

- восстанавливаются мантиссы сомножителей (добавляются единицы целой части);
- после чего мантиссы перемножаются;
- результат нормализуется и, при необходимости, округляется;

- восстанавливаются порядки сомножителей (убираются смещения);
- затем порядки складываются;
- результат сложения порядков смещается;
- знаковые биты перемножаются.

Алгоритм выполнения операций сложения вещественных чисел:

- перед выполнением операций сложения чисел они приводятся к единому (оптимальному) порядку;
- мантиссы восстанавливаются (добавляются единицы целой части);
- восстановленные мантиссы приводятся к новому значению порядка;
- и затем складываются (с учетом их знаков);
- результат нормализуется и, при необходимости, округляется.

3.5 Контрольные задания

Системы счисления:

Задание 3.1: Что называют *системой счисления*? Определите понятия *цифра* и *базисное число*. Почему римскую систему называют *аддитивной*, а десятичную, двоичную и шестнадцатеричную – *аддитивно-мультипликативными* ? Запишите десятичные числа 64 и 587 в римской системе счисления.

Задание 3.2: Определите понятия *основание* и *позиционный вес*, используемые при описании позиционных систем счисления. Запишите десятичные числа 2; 9; 15.125; 128.5; 256.625 и 65625.25 в двоичной, восьмеричной и шестнадцатеричной системах счисления.

Задание 3.3: Запишите двоичные числа 1010; 1111.1; 10000.001 и 11111111.01 в десятичной, восьмеричной и шестнадцатеричной системах счисления.

Задание 3.4: Запишите шестнадцатеричные числа ABCD и 866 в двоичной и двоично-шестнадцатеричной системах счисления. Сколько двоичных разрядов потребуется для записи 8-разрядного шестнадцатеричного числа?

Задание 3.5: Сколько нулей и единиц в записи двоичного числа, соответствующего десятичному числу, вычисленному по формуле: $2^{153} + 2^{53} + 2^3$?

Двоичное кодирование текстовых символов:

Задание 3.6: Экспериментально исследуйте структуру 2-х кодовых таблиц стандарта ASCII (DOS Cp866, Windows Cp1251) и 2-х кодовых таблиц стандарта UniCode (UTF-8 и UTF-16). Для проведения эксперимента используйте приложение Windows *Far Manager* (при его установке проконтролируйте наличие плагина⁵ *FarTrans*, при его отсутствии – установите).

- Используя встроенный текстовый редактор, создайте 4 небольших текстовых файла одинакового содержания (несколько коротких строк текста на русском и английском языках). При создании (Shift+F4) каждого файла выбирайте из меню соответствующую символьную кодировку.
- Переведите редактируемый файл в режим просмотра (F6 – view), установите шестнадцатеричный формат просмотра (F4 – Hex), сравните коды одинаковых символов (в русских и английских строках текста) в разных кодировках.
- Определите управляющие коды, используемые для перевода строки.
- Откройте каждый их файлов стандартным Windows-редактором (например, *Блокнот* или *WordPad*). Прокомментируйте результаты. Какие кодировки символов поддерживаются этими редакторами?

Задание 3.7: * Напишите *Python*-программу для определения кода текстового символа. Используя эту программу, экспериментально определите диапазоны кодов, выделенные следующим группам символов:

- арабским цифрам;
- знакам препинания;
- строчным и прописным буквам английского и русского алфавитов.

Эксперимент проведите для любых двух кодовых таблиц, поддерживаемых языком *Python*.

Задание 3.8: * Предложите алгоритм преобразования строчных букв в прописные и обратно. Напишите программу (функцию), принимающую символ и печатающую этот символ и его же в противоположном регистре.

⁵ Для тех, кто еще не знаком с этим термином, следует пояснить, что *плагин* (от англ. *to plug in* – вставить вилку в розетку, подключить) – это отдельно скомпилированный программный модуль, снабженный простым интерфейсом «подключения» к основному программному приложению и предназначенный, как правило, для расширения его функциональных возможностей. Плагин *FarTrans* расширяет возможности приложения *Far Manager* по работе с кодовыми страницами. Для установки плагина следуйте инструкциям разработчика.

Задание 3.9: * Глубоко законспирированный агент Юстас, нелегально работающий на территории стратегического противника, регулярно отправляет в Центр текстовые сообщения (на русском и/или английском языках) и в ответ получает из Центра от своего куратора благодарности и инструкции. Все сообщения передаются в зашифрованном виде, причем ключ шифрования регулярно меняется, но всегда имеется и у Юстаса, и у Центра. В качестве ключа используется целое число, которое при шифровании *прибавляется* к коду каждого *буквенного* символа.

Напишите *Python*-программы для шифрования и дешифрования сообщений, помня о том, что в одном сообщении могут быть символы как русских, так и английских алфавитов, а также о том, что и Юстас, и его куратор из Центра – высокообразованные люди, регулярно (и правильно) использующие в своих сообщениях как строчные, так и прописные буквы. В процессе шифрования / дешифрования буквенных символов должен быть сохранен алфавит (русская или английская буквы) и регистр (строчная или прописная буквы).

Рекомендация: при написании программы используйте информацию о структуре кодовых таблиц (7-е задание).

9.1 Простой ключ: в качестве ключа *Key* используется целое число (в диапазоне от -128 до $+128$), одинаковое для всех символов шифруемого текста.

9.2 Сложный ключ: для каждого *i*-того символа шифруемого текста используется индивидуальный ключ *Key[i]*, значение которого совпадает с кодом *Code[i]* шифруемого символа.

Задание 3.10: Список фамилий русскоязычных студентов, представленный в кодировке Windows Cp1251, программно сортируется в алфавитном порядке (по возрастанию). В списке допущены досадные опечатки при написании фамилий двух студентов: *Афанасьев* записан, как *афанасьев*, а *Африканов* – как *аФриканов*.

- Как изменится положение фамилий этих пострадавших студентов в отсортированном списке при условии, что все остальные студенты именованы в строгом соответствии с правилами записи имен-собственных?
- Насколько далеко в отсортированном списке окажутся фамилии этих двух студентов от фамилий студенток *Арбузовой* и *Ягодкиной*?

Задание 3.11: Студентка *Арбузова* (из предыдущего задания), выйдя замуж за иностранца, сменила фамилию на *Watermelon's*, и это важное в ее жизни событие было официально зарегистрировано в списке студентов. При редактировании списка были также обнаружены и исправлены опечатки в фамилиях студентов *Афанасьева* и *Африканова*, после чего список был заново отсортирован в алфавитном порядке. Как изменилось положение фамилии этой (бывшей *Арбузовой*) студентки в обновленном списке?

Задание 3.12: * Правильность ответов на вопросы 10-го и 11-го заданий подтвердите экспериментально, составив соответствующую *Python*-программу.

Двоичное кодирование целых десятичных чисел:

Задание 3.13: Определите диапазон допустимых значений переменной 2 байтового типа *Unsigned Integer*. Ответ представьте в двоичной, шестнадцатеричной и десятичной системах счисления.

Задание 3.14: Определите диапазон допустимых значений переменной 2 байтового типа *Integer*. Ответ представьте в двоичной, шестнадцатеричной и десятичной системах счисления.

Задание 3.15: Дайте определения прямому, обратному и дополнительному кодам n -разрядного двоичного числа. Приведите алгоритм формирования дополнительного кода числа по его прямому коду.

Задание 3.16: Переменная типа *Integer*, для хранения которой выделено машинное слово длиной в 4 байта, последовательно получила значения 32768 и -32768 . Какими двоичными числами представлены эти значения в ячейке памяти?

Задание 3.17: Используя операцию двоичной инверсии (\sim) языка *Python*, экспериментально подтвердите (или опровергните) гипотезу о том, что положительные целые числа хранятся в прямом двоичном коде с нулевым знаковым битом, а отрицательные – в дополнительном коде с единичным знаковым битом.

Задание 3.18: * Напишите *Python*-программу перевода прямого n -разрядного двоичного кода в дополнительный код и обратно.

Двоичное кодирование вещественных десятичных чисел:

Задание 3.19: Дайте определения понятий «мантисса», «порядок», «нормализованная мантисса» и «денормализованная мантисса» вещественного числа, заданного в некоторой позиционной системе счисления.

Задание 3.20: Машинные слова $3FA00000_{16}$, $BF800000_{16}$ и 41820000_{16} представляют различные значения переменной типа `float` (*binary-32*). Каким десятичным числам соответствуют эти значения переменной ?

Задание 3.21: Переменная типа `float` последовательно получила значения «16,5», «-1,25», и «-8,625». Как эти значения будут представлены в памяти компьютера при ее стандартном кодировании с точностью *Single Precision* (*binary-32*) и с точностью *Half Precision* (*binary-16*)?

Задание 3.22: Оцените (приблизительно) диапазон допустимых значений и максимальную точность представления (десятичных знаков после запятой) переменной стандартного типа *Half Precision* (*binary-16*).

Задание 3.23: * Напишите *Python*-программу нормализации вещественного числа: пользователь вводит целое число – основание системы счисления и затем цифровую строку – запись какого-то вещественного числа в этой системе счисления с запятой в качестве разделителя целой и дробной частей числа. Программа выводит мантиссу (цифровую строку) и порядок нормализованного числа.

Задание 3.24: * Напишите *Python*-программу интерпретации двоичного кода вещественного десятичного числа, заданного в формате *binary-16* стандарта IEEE-754. Пользователь вводит цифровую строку, состоящую из 16 нулей и единиц в произвольном порядке, программа выводит десятичное значение закодированного числа.

ГЛАВА 4

АППАРАТНОЕ ОБЕСПЕЧЕНИЕ ЭВМ

Компьютерная программа – это языковое описание алгоритма решения задачи, и если алгоритм описан на языке высокого уровня, то программа (так называемый *исходный код*) – не более, чем текст, содержащий описание последовательности выполнения алгоритмических операций. Такую программу невозможно выполнить и, следовательно, нельзя ощутить какую-либо ее полезность для потребителя (исключая, разумеется, полезность для тренировки ума читателя исходного кода или доход от использования интеллектуальной собственности для ее разработчика).

Для того, чтобы даже широко востребованная и хорошо написанная компьютерная программа могла стать практически полезной, необходимы, как минимум, два условия: во-первых, программа должна быть представлена *исполнимым кодом*, то есть должна быть написана на *языке низкого уровня* – языке машинных команд, воспринимаемых аппаратными устройствами, и, во-вторых, необходима аппаратура – комплекс электронно-вычислительных устройств, способных выполнять машинные команды.

4.1 Базовые принципы функционирования ЭВМ

Базовые принципы функционирования ЭВМ были сформулированы американским ученым и инженером фон Нейманом⁶ в 1946 году. Эти принципы были выработаны в рамках проекта вычислительной машины EDVAC – одной из первых ЭВМ, использующих программу, хранимую во внутренней памяти, а не на внешнем устройстве (например, на наборной электрической панели, как это было, у предшественника EDVAC – машины ENIAC, перепрограммирование которой требовало длительного и трудоемкого процесса переустановки перемычек).

1-й принцип – определяет структуру ЭВМ и минимально необходимый состав ее функциональных компонентов (рисунок 4.1): арифметико-логическое устройство (АЛУ), запоминающее устройство (ЗУ), устройства ввода и вывода (УВВ) и устройство управления (УУ).

⁶ Эти принципы, впоследствии названные «*принципами фон Неймана*», впервые были опубликованы в статье «Предварительное рассмотрение логического конструирования электронного вычислительного устройства», написанной Джоном фон Нейманом в соавторстве с Артуром Бёрксом и Германом Голдстайном.

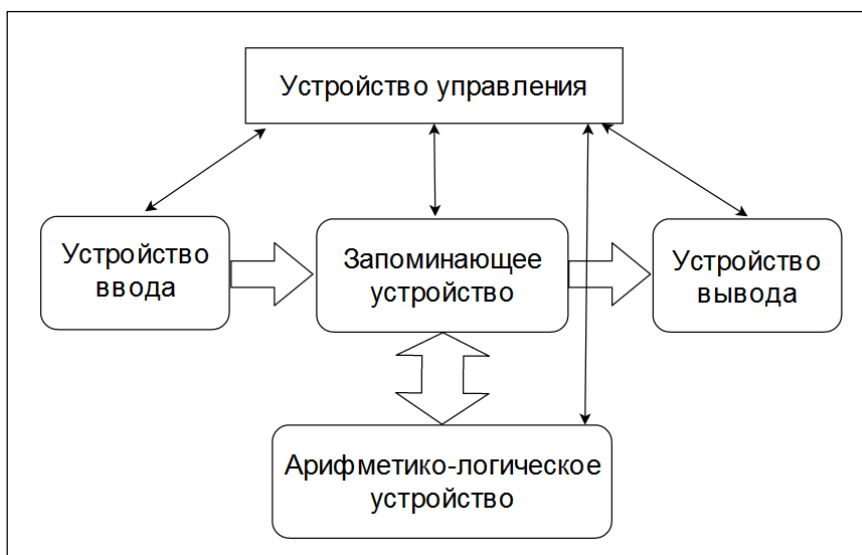


Рисунок 4.1 – Обобщенная структура фон-неймановской ЭВМ

АЛУ – это основное операционное устройство ЭВМ, производящее арифметическую и логическую обработку данных.

ЗУ – предназначено для хранения данных и машинных программ.

УВВ – обеспечивают ввод информации от устройств подготовки данных и вывод результатов вычислений на устройства отображения информации.

УУ – обрабатывает информацию о состоянии всех прочих устройств и вырабатывает сигналы управления этими устройствами.

Одной из основных функций УУ является вычисление адресов ячеек памяти, из которых читается или в которые записывается информация, обрабатываемая в АЛУ. В современных компьютерах АЛУ и УУ вместе обозначают термином *центральный процессор*.

2-й принцип – принцип двоичного кодирования: любая информация, обрабатываемая в ЭВМ, представлена в числовом двоичном коде и разделена на единицы, называемые машинными словами. Машинное слово – это единица данных, обрабатываемая в АЛУ за один цикл. Длина машинного слова (разрядность двоичного кода) – одна из основных характеристик АЛУ. В первых персональных компьютерах длина машинного слова составляла 8 или 16 битов.

3-й принцип – принцип адресуемости памяти. Согласно этому принципу, запоминающее устройство состоит из последовательности пронумерованных ячеек, при этом номер ячейки используется в качестве ее адреса, указывающего на расположение ячейки в адресном пространстве компьютера. Реализация доступа к памяти по адресам ее ячеек дала возможность использования в программировании переменных, которые, по существу, выполняют роль именованных ссылок на определенные ячейки памяти.

4-й принцип – принцип однородности памяти: и программа, и обрабатываемые этой программой данные хранятся в едином запоминающем устройстве в виде последовательности машинных слов.

Алгоритм обработки данных представляется в памяти компьютера последовательностью *машинных команд*, каждая из которых задает некоторую операцию над данными. Множество операций, выполняемых АЛУ, называется *системой команд*⁷. Машинная команда содержит *код операции* и *адресные поля*⁸: код операции указывает на действие, которое следует выполнить в АЛУ над операндами команды, а адресные поля могут содержать значения этих операндов или адреса ячеек памяти, выделенные для операндов или результатов операции.

Множество взаимосвязанных машинных команд, совместно реализующих какой-либо алгоритм обработки данных, называется *машинной программой*. Уже давно никто не занимается программированием на языке машинных команд – машинная программа формируется специальной программой-транслятором⁹, которая преобразует *исходный код* программы, написанной программистом на языке высокого уровня, в *исполнимый код* программы на языке машинных команд соответствующего процессора.

У принципа однородности памяти имеется два важных последствия:

- 1) Над машинными командами, при необходимости, можно выполнять те же действия, что и над данными.
- 2) Только программа может определить, какая информация закодирована в определенном машинном слове, и, следовательно, только программа может интерпретировать содержимое этого машинного слова.

Хранение программы в памяти ЭВМ с возможностью программной модификации машинных команд создало основу программирования в его современном понимании.

⁷ Система команд микропроцессора *Intel 8086*, на базе которого строились первые IBM-совместимые ПК, включала 98 команд длиной от одного до четырех байтов, системы команд современных микропроцессоров существенно богаче.

⁸ Имеются и *безадресные* команды – в этом случае адреса операндов команды считаются заданными по умолчанию (как правило, это адреса внутренних регистров процессора).

⁹ Программа-транслятор тоже должна быть написана программистом на каком-либо языке и затем транслирована в машинный код. Написание трансляторов – узкоспециальная область системного программирования, в которой используются низкоуровневые машинно-ориентированные языки или низкоуровневые возможности, имеющиеся у некоторых языков программирования высокого уровня.

5-й принцип – принцип программного управления процессом выполнения самой машинной программы.

АЛУ последовательно выполняет машинную программу, предварительно записанную в память с помощью устройства ввода, последовательно считывая из памяти очередные машинные команды, начиная с ее первой команды (адрес которой предполагается известным). При этом, в зависимости от результата выполнения текущей команды, управляющее устройство автоматически вычисляет адрес машинного слова, содержащего следующую команду программы – это может быть или адрес следующей ячейки, если выполняется линейный участок программы, или адрес какой-либо другой ячейки, на который указывает команда передачи управления (команда перехода).

Машинная команда, в соответствии с ее «кодом операции», может инициировать выполнение АЛУ следующих действий:

- ввод данных из внешнего устройства в ячейки памяти;
- чтение из ячеек памяти данных, используемых в качестве операндов арифметических или логических операций;
- выполнение логических или арифметических операций;
- запись результатов операции в ячейку памяти;
- вывод данных из памяти на внешнее устройство.

Программа закончит свою работу в результате выполнения специальной команды, которая будет интерпретирована, как «конец программы».

4.2 Типовая архитектура простейшей ЭВМ

Персональные компьютеры построены на базе простейшей *магистральной* архитектуры (рисунок 4.2), называемой также архитектурой «с общей шиной». Все модули компьютера (модули памяти и периферийные устройства) подключены к общей магистрали (системной шине), обеспечивающей их взаимодействие путем обмена данными, адресной и управляющей информацией.

Магистральная архитектура – самая дешевая в реализации, что существенно при массовом производстве ПК, но далеко не самая эффективная по критерию производительности обмена данными, так как временной ресурс общей шины физически делят между собой три логически автономных канала, обеспечивающих прием и передачу информации между центральным процессором, модулями памяти и периферийными устройствами, а также прием и передачу сигналов, управляющих всеми этими устройствами:

– **Шина адреса (ША)**, по которой центральный процессор передает модулям памяти *адреса ячеек*, из которых требуется прочитать очередную машинную команду или данные для их последующей обработки командой, или в которые требуется записать данные – результаты выполнения команды.

– **Шина данных (ШД)** используется для двусторонней передачи различных данных (машинных команд, их операндов или результатов выполнения) между центральным процессором и остальными модулями.

– **Шина управления (ШУ)**, по которой центральный процессор передает всем остальным модулям управляющие сигналы и/или принимает сигналы об их состоянии. На этой шине основной является *линия операции*, на которую устройство-задатчик передает, и с которой устройство-исполнитель принимает код операции обмена данными (*чтение* или *запись*). Кроме линии операции, шина управления включает *линию занятости* и *линию синхронизации*.

Периферийные устройства подключаются к системной шине через специальные программируемые устройства (называемые адаптерами или контроллерами), обеспечивающие интерфейсы взаимодействия с внешним оборудованием. Среди них есть универсальные, например, последовательные (СОМ), параллельные (LPT), и специальные, например, контроллеры дисковых запоминающих устройств, видео-адаптеры, контроллер прерываний, обслуживающий внешние устройства, взаимодействующих с системой в асинхронном временном режиме, или контроллер прямого доступа к памяти.

Каждый контроллер имеет собственный набор встроенных регистров, называемых *портами ввода/вывода* и связанных с системной шиной компьютера. Различают *порты данных* (или *буферные порты*), предназначенные для накопления (буферизации) данных при их приеме/передаче, *порты состояния*, используемые для хранения информации о состоянии контроллера или сопряженного с ним периферийного устройства, и *порты управления*, используемые для получения команд от центрального процессора.

Информация, передаваемая по системной шине, представлена в числовой форме (вспомним 2-й принцип фон-Неймана) – в параллельном двоичном коде, что обеспечивает возможность передачи за один цикл двоичного числа, разрядность которого не превышает разрядности соответствующей шины. От характеристик системной шины зависят объем памяти компьютера и максимальное количество подключаемых периферийных устройств.

Разрядность *шины данных* должна соответствовать разрядности центрального процессора, то есть длине машинного слова, обрабатываемого в АЛУ за один цикл. Если, например, 8-разрядный процессор «работает» в компьютере с 16-разрядной шиной данных, то такая разрядность шины явно избыточна, так как ее старшие 8 битов вряд ли когда-либо будут использоваться. Противоположная ситуация (например, 32-разрядный процессор и 16-разрядная шина данных) может привести к двукратной потере производительности вычислений, так как «доставка» процессору 4-байтового операнда для выполнения какой-либо операции потребует двукратного обращения к памяти.

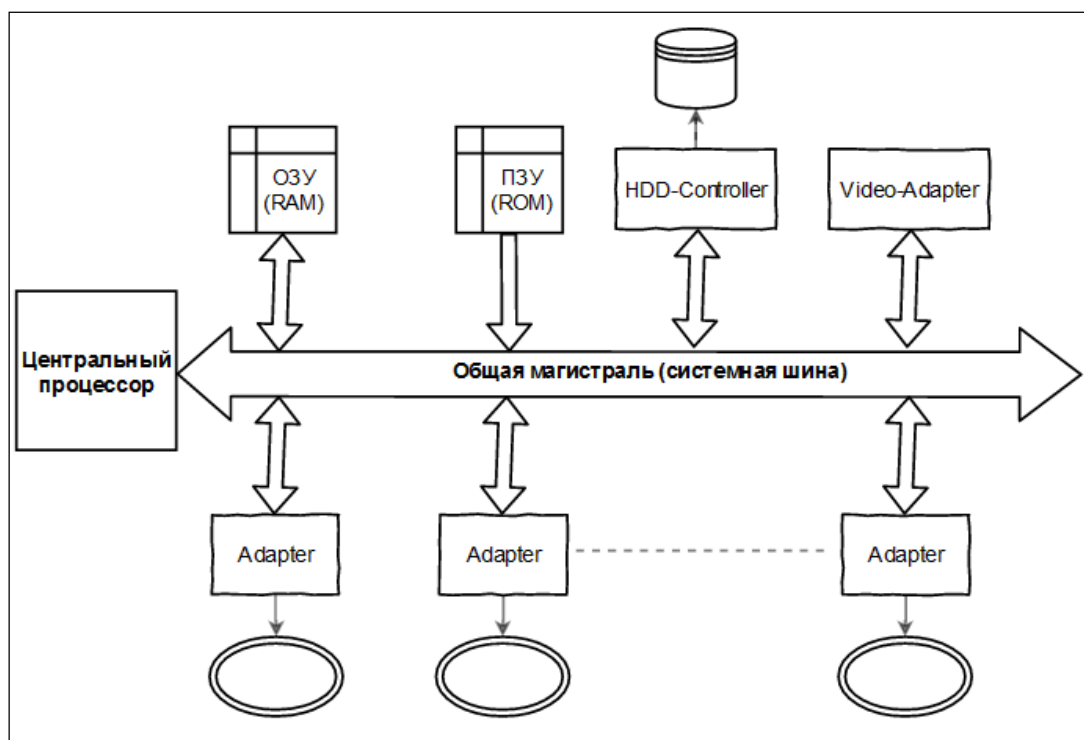


Рисунок 4.2 – Магистральная архитектура ЭВМ

Минимальная адресуемая ячейка памяти имеет размер в 1 байт. Разрядность *адресной шины* определяет максимально возможное количество однобайтовых ячеек памяти, к которым процессор может обратиться для чтения или записи, передав по этой шине двоичное число, представляющее адрес (порядковый номер) соответствующей ячейки.

В компьютере с n -разрядной адресной шиной можно использовать до 2^n ячеек памяти – с адресами от 0 до $2^n - 1$. Например, 8-разрядная адресная шина обеспечит доступ к памяти объемом не больше 256 байтов, 16-разрядная – уже 65536 (64 Кб), а 20-разрядная – 1 Мб. Заметим, что 1 дополнительный бит адресной шины удваивает предельный размер адресного пространства.

Для иллюстрации и исследования алгоритмов взаимодействия программных и аппаратных компонентов компьютера в нашей дисциплине будет использоваться виртуальная «DOS-машина» (аналог IBM-совместимого ПК 1983 года выпуска), работающая под управлением операционной системы MS DOS и построенная на базе 16-разрядного микропроцессора Intel 8086 (рисунок 4.3). АЛУ этого процессора способно обрабатывать за один цикл машинные слова разрядностью не более 16 бит, и все его внутренние регистры, в том числе и адресные, также 16-разрядные. При этом в конструкции процессора предусмотрено специальное схемотехническое решение – *сумматор адреса*, обеспечивающий возможность работы с 20-разрядной адресной шиной, что позволило 16-кратно (до 1 Мб) увеличить объем адресного пространства.

Отметим еще одну особенность системы адресации, реализованную в IBM-совместимых ПК. Адаптеры, через которые периферийные устройства подключены к системной шине (рисунок 4.2), имеют собственные «ячейки памяти» – последовательно пронумерованные однобайтовые регистры (называемые *портами ввода-вывода*), через которые центральный процессор производит обмен данными с периферийными устройствами. Номер порта – это его адрес в адресном пространстве ввода-вывода (п. 4.6.2), по которому процессор получает доступ к порту через общую или автономную адресную шину, аналогично тому, как это делается для ячеек модулей памяти.

4.3 Центральный процессор

На рисунке 4.3 представлена упрощенная структурная схема микропроцессора *Intel-8086*. Этот микропроцессор – 16-разрядный, то есть АЛУ процессора способно обрабатывать за один цикл двух-байтовые машинные слова, и все регистры этого микропроцессора – также 16-разрядные. В структуре микропроцессора можно выделить три основных функциональных блока:

- *устройство шинного интерфейса*, обеспечивающее взаимодействие с шиной адреса / данных (ШАД) и выборку из памяти очередных машинных команд и их операндов;
- *операционное устройство*, исполняющее эти команды;
- *устройство управления*, получающее информацию о текущем состоянии блоков процессора и внешнего оборудования и синхронизирующее процессы их взаимодействия.

Блоки могут работать параллельно, совмещая во времени процессы выборки команд из памяти и их исполнения в операционном устройстве.

Шинный интерфейс обеспечивает взаимодействие микропроцессора с системной шиной компьютера: формирует физические адреса ячеек памяти и портов ввода-вывода, производит выборку данных из этих ячеек (машинных команд и/или их операндов), формирует очередь команд и записывает результаты их выполнения в соответствующие ячейки памяти или порты ввода-вывода.

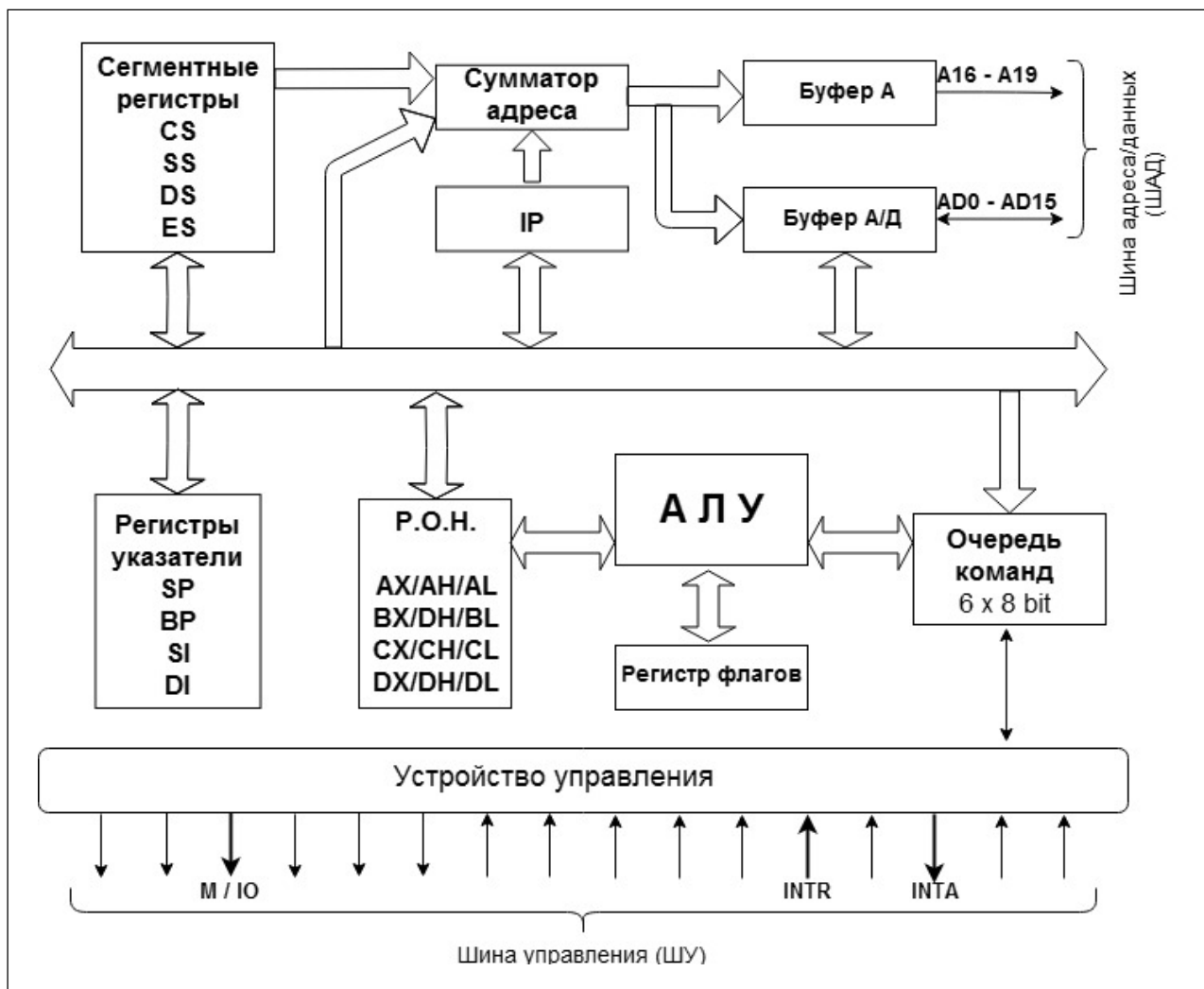


Рисунок 4.3 – Схема 16-разрядного микропроцессора Intel-8086

В состав шинного интерфейса входят:

- *Очередь команд* – регистровая память объемом 6 байт, предназначенная для временного хранения выбранных из памяти команд, ожидающих исполнения в операционном устройстве. Очередь может содержать от одной до шести машинных команд (в зависимости от их суммарной длины) и обслуживается по конвейерной схеме в порядке поступления команд (*FIFO – First-Input-First-Output*).

- *Модуль формирования физического адреса*, включающий:
 - четыре сегментных регистра, содержащих *номера сегментов памяти*, выделяемых машинным программам при их выполнении:
 - регистр **CS** (*Code Segment*) содержит номер сегмента памяти, используемого для хранения программного кода;
 - регистр **DS** (*Data Segment*) содержит номер сегмента памяти, используемого для хранения данных, обрабатываемых программой;
 - регистр **SS** (*Stack Segment*) содержит номер сегмента, в котором организован *стек* – область памяти, обслуживаемая в порядке, обратном порядку поступления в стек данных (*LIFO – Last-Input-First-Output*); стек используется для временного сохранения содержимого внутренних регистров процессора при прерывании выполнения машинной программы (например, при вызове подпрограммы или программы обработки прерывания);
 - регистр **ES** содержит номер дополнительного сегмента данных.
 - указатель команд **IP** (*Instruction Pointer*), содержащий начальный адрес очередной машинной команды (точнее – только часть адреса, а именно, смещение от начала сегмента, номер которого указан в регистре CS). Значение регистра *IP* автоматически инкрементируется (увеличивается на определенное число) после завершения обработки текущей команды.
 - сумматор адреса, выполняющий арифметическую операцию «суммирования со сдвигом» содержимого двух 16-разрядных регистров (например, регистра-указателя *IP* и сегментного регистра *CS*), в результате которой формируется 20-разрядный физический адрес, выставляемый на 20-разрядную адресную системную шину.
 - два *буфера* для временного хранения данных или физического адреса перед их передачей на системную шину адреса/данных (ШАД).

Операционное устройство обеспечивает последовательное выполнение машинных команд, предварительно записанных в очередь команд, и содержит следующие модули:

- *Арифметико-логическое устройство (АЛУ)*, предназначенное для выполнения арифметических и логических операций.

- *Регистры общего назначения* (РОН: *A, B, C* и *D*), иногда называемые «сверх-оперативной памятью» и используемые для временного хранения операндов и промежуточных результатов выполнения команд¹⁰. Эти регистры допускают раздельное использование своих младших (*L* – *Low*) и старших (*H* – *High*) байтов, обеспечивая тем самым возможность хранения и извлечения как 16-разрядных машинных слов (*X*), так и отдельных байтов данных:
 - *AX* (*AL+AH*), называемый *регистром-аккумулятором*, используется в операциях обмена данными с устройствами ввода/вывода (*IN* и *OUT*) и в операциях умножения и деления;
 - *BX* (*BL+BH*), называемый *базовым регистром*, используется в вычислениях адреса, указывая на начальный адрес структуры данных в памяти;
 - *CX* (*CL+CH*) используется в качестве счетчика (*count*) циклов, определяющего количество повторов некоторой операции;
 - *DX* (*DL+DH*) используется для хранения данных (*data*), передаваемые в подпрограммы для обработки.
- *Адресные регистры-указатели* (*SP, BP, SI* и *DI*), используемые совместно с сегментными регистрами при вычислении физического адреса, аналогично тому, как это делается с указателем команды *IP*:
 - *SP* (*Stack Pointer*): указывает на вершину стека, позволяет всегда производить выборку данных, записанных в стек последними;
 - *BP* (*Base Pointer*): указывает на данные, хранимые в стеке, облегчает доступ к данным, передаваемым через стек;
 - *SI* (*Source Index*): индекс источника данных, обычно связан с сегментным регистром *DS*;
 - *DI* (*Destination Index*): индекс назначения, обычно связан с сегментным регистром *ES*.

¹⁰ Программист (точнее – компилятор, генерирующий машинный код из транслируемого исходного кода, написанного программистом) сам решает, как использовать регистры общего назначения, однако каждый из них имеет еще и свое специфическое назначение, и в ряде случаев машинные команды требуют использования строго определенных регистров: так, например, команды *IN* и *OUT* обеспечивают прием/передачу данных между портом ввода-вывода, адрес которого указан в команде, и регистром-аккумулятором.

- *Регистр флагов (F)*, в котором каждый бит (флаг) содержит признак результата выполненных команды или управляющую информацию, используемую при выполнении операций:
 - *ZF (Zero Flag)*: устанавливается равным «1» при нулевом результате операции;
 - *SF (Sign Flag)*: устанавливается равным старшему биту результата выполнения арифметической операции («1» – при отрицательном результате, «0» – при равном или большем нуля);
 - *CF (Carry Flag)*: фиксирует факт переноса единицы из старшего бита в результате выполнения арифметической операции;
 - *OF (Overflow Flag)*: устанавливается в «1» при получении результата вне допустимого диапазона чисел;
 - *PF (Parity Flag)*: устанавливается в «1», если младший байт результата операции содержат четное число единиц;
 - *IF (Interrupt-enable Flag)*: если флаг установлен в «1», аппаратное прерывание разрешено, если в «0» – прерывание игнорируется;
 - *DF (Direction Flag)*: применяется в командах обработки последовательности байтов в памяти: если флаг установлен в «0», последовательность обрабатывается от младшего адреса к старшему, в противном случае – наоборот;
 - *TF (Trap Flag)*: единичное значение флага устанавливает пошаговый (отладочный) режим выполнения программы, вырабатывая соответствующее прерывание после завершения выполнения каждой машинной команды.

Устройство управления выполняет дешифрацию команды и вырабатывает соответствующие управляющие сигналы, синхронизирующие работу всех блоков микропроцессора, а также обеспечивает взаимодействие процессора с периферийными устройствами через системную шину управления.

4.4 Запоминающие устройства

Согласно одному из принципов фон-Неймана, запоминающее устройство компьютера определено как техническое средство хранения машинных программ и обрабатываемых ими данных, что и определяет основные требования, предъявляемые к таким устройствам:

- большая емкость накопителя, позволяющая хранить сложные программы и большие массивы обрабатываемых ими данных и результатов обработки;
- малое время доступа к памяти для чтения и записи данных, что должно обеспечить высокопроизводительную работу компьютера;
- низкое энергопотребление, а в идеале – полная энергонезависимость;
- малые габаритные размеры;
- разумно низкая стоимость запоминающего устройства.

Идеальный вариант – это когда малогабаритный персональный компьютер укомплектован высокопроизводительной памятью гигантской емкости, информация в которой сохраняется неограниченно долго при выключенном питании, и при этом стоимость запоминающего устройства не выходит за пределы, ограничивающие возможность массовых продаж компьютерной техники.

Очевидно, что в ближайшей перспективе такой вариант так и останется идеальным и практически не достижимым – объективные технические, технологические и финансовые ограничения не позволяют выполнить все эти, часто взаимоисключающие, требования в устройстве какого-то одного типа. Также очевидно и решение проблемы – вместо одного универсального запоминающего устройства компьютер комплектуется несколькими специализированными модулями памяти, каждый из которых предназначен для решения задач определенного класса и удовлетворяет соответствующим требованиям.

Изучение физических принципов работы запоминающих устройств, их возможностей и областей эффективного применения выходит за рамки вводного курса информатики – эти вопросы детально рассматриваются в соответствующих разделах физики, электроники и схемотехники. Историю и перспективы развития технологий хранения информации также оставим за пределами рассмотрения в настоящем учебном пособии, приведем лишь не очень строгую классификацию (рисунок 4.4) запоминающих устройств по их функциональному назначению и специфике использования в программно-аппаратном комплексе ЭВМ.

Деление запоминающих устройств на «основную» и «внешнюю» память отражает различные способы адресации к этим устройствам: ячейки основной памяти расположены в основном адресном пространстве (п. 4.6.1), и доступ к ним осуществляется по системной адресной шине, а внешние запоминающие устройства доступны по шине ввода-вывода (п. 4.6.2) подобно тому, как это делается для прочих периферийных устройств.

Основная память компьютера представлена *оперативным запоминающим устройством (ОЗУ)*, доступ к которому возможен как для чтения, так и для записи данных, что и подчеркивается англоязычным названием таких устройств – *Random Access Memory (RAM)*, и *постоянным запоминающим устройством (ПЗУ)*, доступным центральному процессору только для чтения (*Read Only Memory – ROM*).

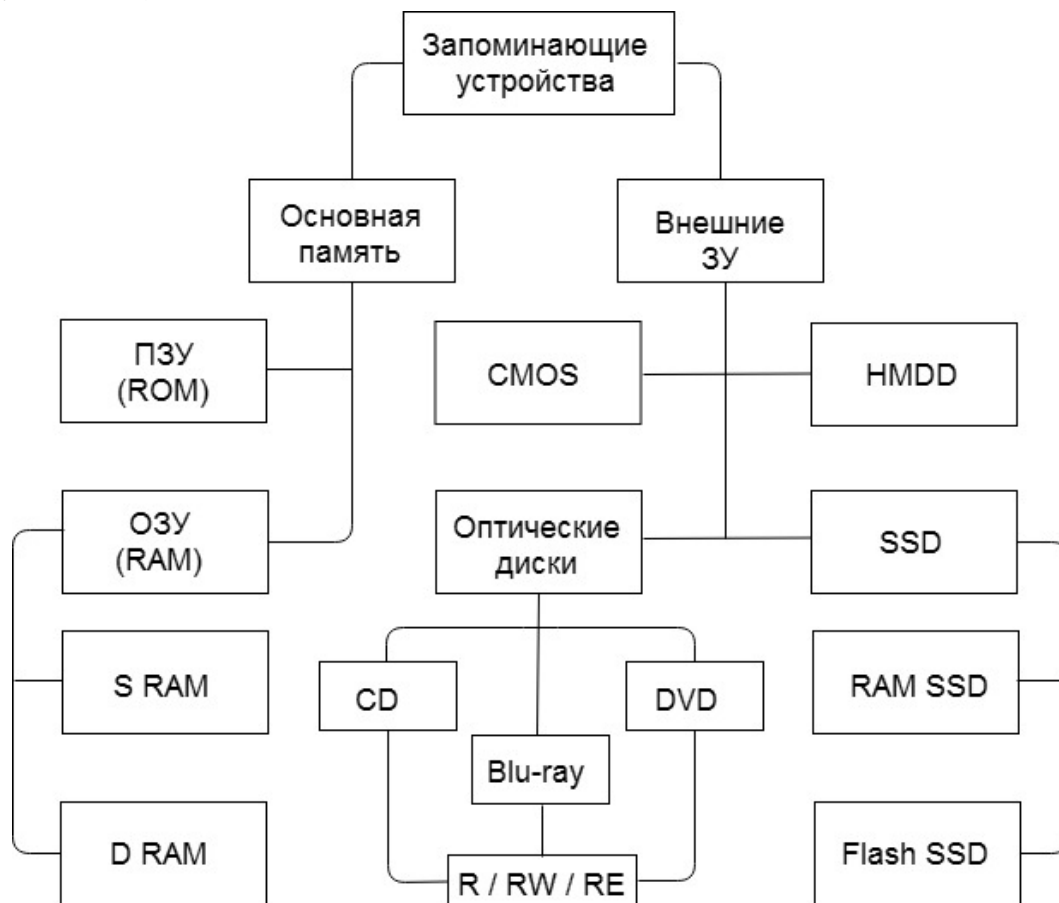


Рисунок 4.4 – Классификация запоминающих устройств по функциональному назначению

Постоянное запоминающее устройство (ПЗУ) – это энергонезависимая память, не требующая затрат энергии для хранения данных. Информация, записанная в ПЗУ, сохраняется в нем после выключения питания компьютера.

ПЗУ программируется («прошивается») при изготовлении микросхемы и предназначено для хранения неизменяемых в процессе работы компьютера программ и данных: постоянные программные компоненты базовой системы ввода-вывода (*ROM BIOS*) и обслуживающие их и структуры данных; программа *POST (Power On Self Testing)*, автоматически запускаемая при включении питания и тестирующая память и подключенное внешнее оборудование; программа начальной загрузки операционной системы.

ПЗУ – относительно медленное устройство, поэтому, если требуется оперативный доступ к данным, записанным в ПЗУ, операционная система копирует эти данные в оперативную память, работающую существенно быстрее.

Оперативное запоминающее устройство (ОЗУ), как следует из его названия, используется в оперативном режиме – для записи и чтения машинных программ и обрабатываемых ими данных. Часть ОЗУ резервируется операционной системой для хранения системного программного обеспечения и данных справочного характера (таблица 4.2), другая часть отдана в распоряжение прикладному программному обеспечению.

ОЗУ компьютера может быть построено на базе модулей динамической (*Dynamic RAM*) или статической (*Static RAM*) памяти.

В основе *DRAM* – схема, состоящая из транзистора и конденсатора, которая обеспечивает хранение одного бита информации: для установки бита в единичное состояние требуется зарядить конденсатор, а для сброса в ноль – соответственно, разрядить. Преимуществом *DRAM* является компактность и дешевизна технической реализации, а недостатком – относительно низкое быстродействие, так как, во-первых, требуется время на зарядку/разрядку конденсаторов, а во-вторых, для поддержания заряженного состояния конденсатора требуется его периодическая «подзарядка». Контроллер модуля *DRAM* периодически приостанавливает операции доступа к памяти для регенерации её содержимого.

Статическая память *SRAM* собрана на триггерах и существенно выигрывает у *DRAM* по производительности: во-первых, изменение напряжения на входе триггера приводит к изменению его состояния практически без задержки, и, во-вторых, отсутствие в схеме *SRAM* конденсатора исключает необходимость периодической регенерации памяти. Недостатком *SRAM* является его высокая стоимость и низкая (по сравнению с *DRAM*) информационная плотность.

Обычно ОЗУ персональных компьютеров строят на базе модулей динамической регенерируемой памяти *DRAM*, а статическую память *SRAM* используют, например, в кэш-памяти микропроцессоров.

Внешние запоминающие устройства (ВЗУ) персональных компьютеров представлены большим разнообразием типов устройств и их конструктивных исполнений. Общее свойство всех типов ВЗУ – энергонезависимость, что, собственно, и позволяет таким устройствам выполнять свое главное предназначение – долговременное и надежное хранение больших объемов информации.

ВЗУ выигрывают у ОЗУ и ПЗУ по такому параметру, как стоимость хранения единицы информации, но существенно проигрывают им по производительности. Последнее не так критично, так как режим доступа к ВЗУ предполагает эпизодическое обращение к ним для загрузки в ОЗУ фрагментов файлов или, наоборот, для записи в файлы содержимого оперативной памяти.

Наиболее распространенным типом ВЗУ в большинстве персональных компьютеров является *накопитель на жёстких магнитных дисках (НЖМД или HMDD – Hard Magnetic Disk Drive)*, основанный на принципе электромагнитной записи. Является *RAM*-устройством, т.е. позволяет производить как чтение, так и запись данных.

HMDD – технически сложное электромеханическое устройство, управляемое специализированным контроллером. Основу конструкции *HMDD* составляет пакет металлических дисков, закрепленных на общем валу, вращающемся с высокой скоростью. Рабочие поверхности дисков (*sides*) покрыты магнито-жестким ферромагнитным сплавом, важным свойством которого является сохранение состояния намагниченности участков поверхности после снятия напряженности магнитного поля. Высокий или низкий уровень намагниченности элементов рабочей поверхности диска используется для двоичного кодирования записанной информации.

Источником магнитного поля при записи информации на диск является магнитная головка (*head*), она же является и приемником электромагнитного сигнала от намагниченной области поверхности диска при чтении информации. Блок магнитных головок (по одной головке на каждую рабочую поверхность диска) совершает возвратно-поступательное перемещение в радиальном направлении под управлением прецизионного дискретного электропривода, обеспечивающего точное позиционирование блока на заданном радиусе. При этом механический контакт магнитной головки с рабочей поверхностью вращающегося диска отсутствует – между ними поддерживается постоянный воздушный зазор величиной в несколько нанометров.

Блок магнитных головок имеет несколько фиксированных радиальных позиций, каждая головка «описывает» на соответствующей рабочей поверхности диска множество концентрично расположенных кольцевых поверхностей, называемых «дорожками» или «треками» (*track*). Множество треков имеющих одинаковый радиус, но расположенных на различных рабочих поверхностях, называются «цилиндром» (*cylinder*).

Каждый трек разделен на множество секторов (*sectors*), при стандартном форматировании количество секторов на всех треках одинаково, одинакова также и информационная емкость сектора – обычно она составляет 512 байтов.

Сектор размером 1/2 Кб является минимально адресуемой «ячейкой» дискового накопителя (для сравнения – минимально адресуемая ячейка ОЗУ и ПЗУ имеет размер в 1 байт), а его информационная емкость M зависит от количества магнитных головок (рабочих поверхностей) и плотности магнитной записи, определяемой количеством дорожек на рабочей поверхности и количеством секторов на дорожке: $M = Heads \times Tracks \times Sectors \times 512$.

Основные характеристики серийно выпускаемых жестких дисков:

- *Форм-фактор*. Включает физический размер (диаметр диска, обычно равный 3.5 или 2.5 дюйма, реже – 1.8, 1.3, 1 и 0,85 дюйма), количество головок / рабочих поверхностей диска (*heads / sides*) и параметры плотности магнитной записи (*cylinders / tracks* и *sectors*).

- *Скорость вращения (spindle speed)* – угловая скорость вращения пакета дисков, измеряемая в оборотах в минуту и влияющая на время доступа к данным, хранимым на *HMDD*. Стандарты скоростей вращения: для ноутбуков – 4200, 5400 и 7200 об/мин.; для ПК – 5400, 7200 и 10 000 об/мин.; для серверов и высокопроизводительных рабочих станций – 10 000 и 15 000 об/мин.

- *Время доступа (random access time)*. Определяет время гарантированного выполнения операции чтения или записи; у разных накопителей может составлять от 2,5 до 16 мс.

- *Ёмкость (capacity)*. Определяет максимальный объем хранимых данных, может достигать нескольких терабайт.

- *Интерфейс*. Определяет правила (протокол) обмена данными. Используются интерфейсы ATA, SATA, SCSI, SAS, FireWire, USB, SDIO, Fibre Channel.

Следующая категория внешних запоминающих устройств – это **оптические диски (optical disc)** – оптико-механические устройства в форме многослойного диска: основа диска изготовлена из поликарбоната, на неё методом напыления нанесены тонкие светоотражающие слои, поверх которых нанесены слои, чувствительные к воздействию луча лазера, и все это покрыто слоем прозрачного защитного материала.

Кодирование информации при ее записи на оптический диск происходит за счет изменения оптических свойств рабочего слоя диска, на котором под воз-

действием лазерного луча формируются мельчайшие выемки, называемые *питами* (*pit*). При считывании данных интенсивность отраженного луча изменяется при попадании на очередной пит, что позволяет декодировать записанную информацию. Множество питов образуют на поверхности диска спиральную дорожку (а не множество концентрических дорожек, как это имеет место в магнитных дисках).

Используются различные технологии и форматы записи информации на оптические носители. Первыми появились CD-R (compact disk) – ROM-устройства, предназначенные только для чтения однократно записанной на нем информации объемом, не превышающим 750 – 800 Мб. Позднее были разработаны технологии многократной перезаписи информации на оптические диски (CD-RW) – однако, такие диски остаются, по существу, все теми же ROM-устройствами, так как количество циклов перезаписи на них весьма ограничено.

На смену CD пришел формат DVD¹¹, обладающий большей плотностью размещения данных как за счет уменьшения расстояния между дорожками спирали (CD – 1,6 мкм, DVD – 0,74 мкм), так и за счет возможности двухслойной записи. Среди множества типов DVD-дисков наиболее распространенным является DVD-5 (односторонний однослойный диск емкостью 4,7 Гб), а наиболее емким – DVD-18 (двухсторонний двухслойный диск емкостью 17 Гб).

Более совершенный формат многослойной записи *Blu-ray* получил свое название от цвета луча коротковолнового (405 нм) синего лазера, используемого для чтения-записи информации. Диски этого формата (называемые *BD-дисками*) обеспечивают еще более высокую плотность записи и, соответственно, обладают большей информационной емкостью: однослойный BD-диск – 25 Гб, двухслойный – 50 Гб, четырехслойный – 128 Гб. Имеются и реализации BD-дисков еще большей емкости.

Оптические носители информации долгое время являлись самым дешевым средством хранения и распространения программного обеспечения, однако сегодня эти устройства используются все реже, уступив место более современным твердотельным запоминающим устройствам.

SSD (*Solid State Drive*, твердотельная статическая память) – запоминающее устройство, соизмеримое по емкости с традиционными магнитными дисками и

¹¹ Первоначально формат DVD разрабатывался для хранения и комфортного воспроизведения видеофильмов, откуда и его название – Digital Video Disk. Позднее этот формат был оптимизирован и для хранения компьютерной информации, при этом его название DVD сохранилось, но расшифровывается уже как Digital Versatile Disk (Цифровой Универсальный Диск).

многократно превосходящее их по производительности за счет отсутствия движущихся механических узлов.

Накопители *RAM SSD* характеризуются сверхвысокой скоростью чтения и записи информации и используются в качестве ОЗУ для ускорения работы систем управления базами данных и мощных графических станций. *RAM SSD* являются энергозависимыми устройствами, они, как правило, оснащаются аккумуляторами, а более дорогие модели – системами оперативного копирования. Основным недостатком *RAM SSD* является их чрезвычайно высокая стоимость.

Энергонезависимые накопители *Flash SSD* также являются RAM-устройствами, позволяющими как читать, так и записывать информацию. Эти устройства появились относительно недавно, но сегодня они прочно закрепились на рынке сменных запоминающих устройств и уверенно завоевывают рынок стационарных ВЗУ, постепенно вытесняя с него традиционные магнитные дисковые накопители. Преимуществами *Flash SSD* по сравнению с *HMD* являются небольшие размеры, позволяющие использовать их в портативных устройствах, низкое энергопотребление и высокая производительность. Основным недостатком *Flash SSD* является их относительно высокая стоимость, существуют также ограничения по количеству циклов перезаписи (около миллиона циклов в современных устройствах).

Во современных моделях ноутбуков *Flash SSD* часто является основным (и единственным) стационарным ВЗУ, а многие персональные компьютеры комплектуются SSD-памятью в качестве дополнительного внешнего накопителя, на который, как правило, устанавливается системное ПО и пользовательские приложения, требующие интенсивного обмена с внешней памятью.

CMOS-память – это небольшое по емкости энергозависимое ЗУ, конструктивно выполненное на специальных полупроводниковых элементах *CMOS*-структуры (*Complementar Metal Oxide Semiconductor*). Особенности *CMOS*-структуры является относительно низкая производительность, что не является критичным для условий ее использования, и низкое энергопотребление, что гораздо важнее, так как фактически обеспечивает энергонезависимый режим ее работы при условии питания от встроенного аккумулятора небольшой емкости. При включенном питании компьютера от сети аккумулятор заряжается, а при выключенном – позволяет достаточно долго сохранять в актуальном состоянии информацию, записанную в *CMOS*-память.

В *CMOS*-памяти хранится информация о конфигурации компьютера, способах загрузки операционной системы, а также текущие дата и время, которые в

процессе загрузки считываются из CMOS-памяти и записываются в область данных BIOS, организованную в основной оперативной памяти (таблица 4.3).

4.5 Периферийное оборудование

Периферией принято называть все то, что расположено вдали от центра, и в этом смысле периферийными будут считаться все устройства, доступ к которым со стороны центрального процессора осуществляется через дополнительные устройства-адаптеры, подключаемые к системной шине (рисунок 4.2).

К периферийным устройствам относятся внешние ЗУ, рассмотренные в п. 4.4, а также множество различных устройств, предназначенных для ввода, вывода и передачи информации и обеспечивающих коммуникации компьютера с внешним миром.

Известное утверждение о том, что периферийные устройства (в отличие от центральных) можно отключить от компьютера, и он после этого сможет продолжить свою работу, далеко от реальности – если с отсутствием принтера или сканера еще можно как-то смириться, то представить себе работающим современный компьютер, лишенный дискового накопителя, клавиатуры, видеомонитора и сетевой карты, весьма проблематично.

Асинхронный режим взаимодействия центрального процессора с периферийными устройствами обеспечивает контроллер прерываний (п.7.1.3), обслуживающий множество периферийных устройств и составляющий аппаратную основу *системы обработки прерываний*, рассмотренной в 7-й главе учебного пособия.

Алгоритмы взаимодействия центрального процессора ПК с клавиатурой и видеомонитором, а также структуры данных, используемые этими алгоритмами, рассматриваются в п.7.2 и п.7.3 учебного пособия и исследуются при выполнении ряда лабораторных работ (глава 8).

Структура адресного пространства ввода-вывода и способы адресации периферийного оборудования обсуждаются далее в п. 4.6.3.

4.6 Адресное пространство ПК

Минимальный размер автономно адресуемой ячейки памяти равен одному байту, при этом, в зависимости от разрядности центрального процессора, бывает необходимо (и весьма полезно для повышения производительности обработки данных) хранить, записывать и считывать не только однобайтовые ячейки, но и ячейки большего размера – 2, 4 или 8 байтов, называемые *машинными словами*. Машинное слово включает ячейки с соседними адресами.

Возможности передачи много-байтовых машинных слов ограничиваются разрядностью системной шины данных, а максимальное количество прямо адресуемых однобайтовых ячеек N определяется разрядностью n системной адресной шины: $N = 2^n$.

Адресное пространство компьютера – это множество ячеек запоминающих устройств, доступных центральному процессору¹² через системную шину адреса. При этом под «ячейками» здесь понимаются как *ячейки модулей памяти* (оперативной – *RAM*, доступной как для чтения, так и для записи, и постоянной – *ROM*, доступной только для чтения), так и *порты ввода-вывода* – регистры различных контроллеров и адаптеров (рисунок 4.2), связывающих периферийные устройства с системной шиной.

4.6.1 Сегментная организация адресного пространства основной памяти

Микропроцессор Intel 8086 поддерживает простейшую модель сегментирования адресного пространства, позволяющую расширить объем физического адресного пространства и сделать машинные программы независимыми от места их физической загрузки в память компьютера.

Процессор обслуживает совмещенные в системной шине адреса/данных (ШАД) 16-разрядную шину данных (ШД) и 20-разрядную адресную шину (ША), взаимодействуя с ними через два специальных буфера (рисунок 4.3):

- Буфер адреса/данных, связывающий внутреннюю магистраль микропроцессора с линиями (AD0 – AD15) ШАД: все 16 битов ШД и совмещенные с ними 16 младших битов ША;
- буфер адреса связан со старшими битами (A16 – A19) шины адреса.

В условиях, когда все регистры микропроцессора, включая адресные регистры, являются 16-разрядными, сегментация адресного пространства позволила расширить адресную шину на 4 бита и, соответственно, 16-кратно увеличить объем памяти компьютера (с $2^{16} = 64$ Кб до $2^{20} = 1$ Мб).

Адресное пространство основной памяти (рисунок 4.6) представлено множеством последовательно пронумерованных и перекрывающихся сегментов

¹² Существует также возможность прямого взаимодействия периферийных устройств, минуя центральный процессор. Прямой доступ к памяти управляется контроллером *DMA (Direct Memory Access)* и используется для высокоскоростной передачи больших блоков данных между памятью и внешними устройствами без использования регистров процессора для промежуточного хранения данных.

фиксированного размера 64 Кб¹³. Номера сегментов записываются в соответствующие 16-разрядные *сегментные регистры* (рисунок 4.3). Например, номер сегмента, выделенного для хранения кода программы, записывается в регистр *CS*, для хранения данных – в регистр *DS*, а для стека – в регистр *SS*.

16-разрядный адрес ячейки памяти (называемый *эффективным адресом*) задается величиной ее смещения относительно начала некоторого сегмента. Для хранения эффективных адресов в процессоре также предусмотрены соответствующие 16-разрядные адресные регистры-указатели, например:

- регистр *IP*, используемый для хранения эффективных адресов машинных команд, задает смещение начального адреса команды относительно начала сегмента, номер которого записан в регистр *CS*;
- регистр *SI* задает смещение источника данных относительно начала сегмента, номер которого записан в регистр *DS*;
- регистр *SP* указывает на вершину стека, организованного в сегменте памяти, номер которого записан в регистр *SS*.

Формирование 20-разрядного линейного адреса *LA* ячейки памяти иллюстрирует следующая арифметическая модель: на вход *сумматора адреса* подаются два 16-разрядных числа – номер сегмента и эффективный адрес; сумматор выполняет над ними операцию *сложения со сдвигом* по следующей формуле (приведенной для случая вычисления адреса машинной команды):

$$LA = CS \times 10000_{\text{b}} + IP$$

При умножении номера сегмента *CS* на двоичное число 10000_b (операция сдвига на 4 разряда влево) получается 20-разрядный *базовый адрес сегмента* – линейный адрес его начальной ячейки. После сложения базового адреса сегмента с эффективным адресом формируется 20-разрядный линейный адрес, который и передается на системную шину: старшие 4 бита передаются через буфер на линии A16–A19 ША, а младшие 16 битов – через другой буфер на линии AD0 – AD15 ШАД. Таким образом, линейный адрес ячейки памяти определяется номером сегмента, задающим его положение в адресном пространстве, и величиной смещения ячейки от начала этого сегмента.

Учитывая, что размер сегмента равен 64 Кб, а смещение соседних сегментов составляет 16 байтов, сегменты могут пересекаться в адресном пространстве,

¹³ В поздних моделях процессоров x86 появилась возможность управления размерами сегментов в диапазоне от 16 б до 64 Кб.

то есть одна и та же ячейка памяти может «принадлежать» нескольким сегментам. Так, например, положение ячейки с линейным адресом 00040_h (рисунок 4.6) может быть задано смещением 0040_h относительно начала 0-го сегмента, смещением 0030_h относительно начала 1-го сегмента или смещением 0020_h относительно начала 2-го.

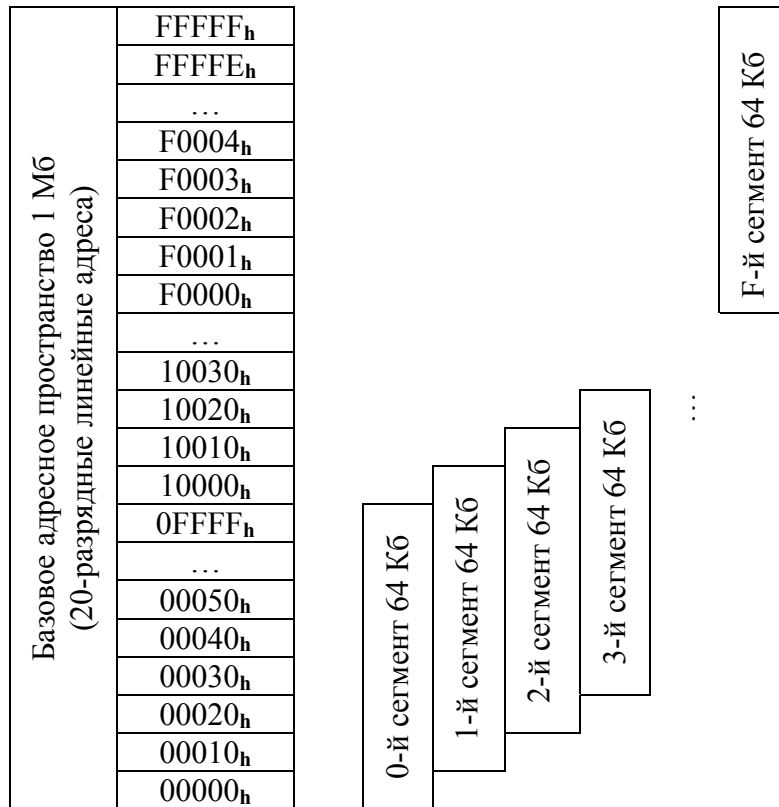


Рисунок 4.6 – Схема сегментации базового адресного пространства

Сегментные адреса принято записывать двумя четырехразрядными шестнадцатеричными числами, разделенными двоеточием: например: в записи адреса [0040:001A]_h слева записан номер сегмента (40_h), а справа – смещение (1A_h) от его начала.

На рисунке 4.7 приведен экранный образ фрагмента оперативной памяти объемом 160 байтов в диапазоне адресов от [0040:0000]_h до [0040:009F]_h, выведенный на экран одной из инструментальных программ-анализаторов памяти.

0040:0000	F8	03	F8	02	E8	03	E8	02	BC	03	78	03	78	02	C0	9F
0040:0010	23	C8	00	80	02	00	00	20	00	00	24	00	24	00	30	52
0040:0020	30	52	34	4B	00	00	00	00	00	00	00	00	00	00	00	00
0040:0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0040:0040	E4	00	C3	00	00	00	00	00	00	03	50	00	00	10	00	00
0040:0050	03	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0040:0060	07	06	00	D4	03	29	30	F8	03	00	F0	04	0F	DF	0C	00
0040:0070	00	00	00	00	00	00	00	00	14	14	14	14	01	01	01	01
0040:0080	1E	00	3E	00	18	10	00	60	09	11	0B	C9	58	01	00	03
0040:0090	07	07	00	00	00	00	10	02	00	00	00	00	00	00	00	00

Рисунок 4.7 – Фрагмент адресного пространства ОЗУ

В левой области экрана отображается сегментный адрес некоторой ячейки памяти, а в правой области – данные, прочитанные программой из этой и последующих 15 ячеек.

Данные отображаются построчно – каждая экранная строка отображает 16-байтовый блок памяти. Так, в однобайтовой ячейке с адресом $[0040:0000]_h$ записано двоичное число, эквивалентное шестнадцатеричному числу $F8_h$, а в ячейке $[0040:008A]_h$ – числу $0B_h$.

За один цикл обмена с памятью по шине данных передается 16-разрядное машинное слово, сформированное из содержимого пары «соседних» однобайтовых ячеек памяти, линейные адреса которых отличаются на 1. При этом в Intel-совместимых компьютерах младшие 8 битов (D0 – D7) машинного слова хранятся в байте с меньшим (четным) номером, а следующий за ним нечетный байт содержит старшие 8 битов (D8 – D15) машинного слова.

На рисунке 4.7 байт с линейным адресом $[00400]_h$ содержит число $F8$, следующий за ним байт с адресом $[00401]_h$ – число 03 , байт $[00402]_h$ – число $F8$, а байт $[00403]_h$ – число 02 . При этом первая пара байтов образует 16-разрядное машинное слово $03F8_h$, а не $F803_h$, как это визуально представлено на экране, а вторая – $02F8_h$, а не $F802_h$, как может показаться человеку, привыкшему читать числа слева направо от старших разрядов к младшим.

Анализируя рассмотренную систему сегментной организации памяти микропроцессора Intel 8086, можно сделать следующие выводы:

- 1) Процессор может поддерживать не более $2^{16} = 65536$ различных сегментов памяти, так как все его сегментные регистры – 16-разрядные.
- 2) Размер сегмента не может превышать $2^{16} = 65536$ байтов (64 Кб), так как регистры, используемые для хранения эффективных адресов (смещений от начала сегментов), также 16-разрядные.

3) Сегменты выровнены по 16-байтовым границам: это, в частности, означает, что если номера сегментов отличаются на единицу, то их базовые адреса отличаются на 16 (или на 10_{h} в шестнадцатеричной системе счисления).

4) Формально существует 4096 альтернативных способов сегментной записи одного и того же адреса, однако, это утверждение будет справедливым только для «срединной части» адресного пространства. Например, два сегментных адреса $[0000:0408]_{\text{h}}$ и $[0040:0008]_{\text{h}}$ определяют один и тот же линейный адрес 00408_{h} , а для линейных адресов $[00001]_{\text{h}}$ и $[FFFFFF]_{\text{h}}$ существует единственный такой способ – соответственно, $[0000:0001]_{\text{h}}$ и $[FFFF:000F]_{\text{h}}$.

5) Используемый способ формирования линейного адреса путем суммирования базового адреса сегмента со смещением позволяет определить линейные адреса, находящиеся за пределами 1-мегабайтной границы. Так, например, сегментный адрес $[FFFF:FFFF]$ соответствует линейному адресу $[10FFFEF]_{\text{h}}$, что требует увеличения на «1» разрядности адресной шины. В процессоре Intel 8086 игнорируется 20-й бит вычисленного линейного адреса, который для рассмотренного выше примера будет иметь значение $[OFFFEF]_{\text{h}}$.

Завершая обсуждение сегментной организации памяти, следует отметить, что в последующих моделях микропроцессоров Intel x86 отпала необходимость «искусственного» расширения адресного пространства за счет ее сегментации: микропроцессор Intel 80386DX, выпущенный в конце 1985 года, уже имел 32-разрядную адресную шину, что обеспечивало возможность прямой адресации к памяти объемом до 4Гб, а с появлением 64-разрядных процессоров размер адресного пространства увеличился (теоретически) до 16 миллионов терабайт.

При этом разработчики процессоров не только не отказались от сегментной модели памяти (хотя и были неудавшиеся попытки такого отказа), но и сделали ее еще более гибкой – например, расширился набор сегментных регистров, появилась возможность работы с сегментами памяти переменной длины.

4.6.2 Стандартное распределение базового адресного пространства

Процессор с 20-разрядной ША обеспечивает размер адресного пространства в 1 Мб с диапазоном линейных адресов однобайтовых ячеек памяти от $[00000]_{\text{h}}$ до $[FFFFFF]_{\text{h}}$. Распоряжается этим адресным пространством операционная система – главная «программа» компьютера, управляющая работой всех его аппаратных компонентов и прикладных пользовательских программ.

Таблица 4.1 иллюстрирует стандартное распределение базового адресного пространства ПК (включая ОЗУ, ПЗУ и видеопамять), построенного на базе микропроцессора Intel 8086.

В таблице 4.2 показано распределение оперативной памяти ПК, а в таблице 4.3 представлена информационная структура *области данных BIOS* – специальной области ОЗУ размером 256 байтов, в которой операционная система хранит служебные структуры данных и адресную информацию, используемую в операциях ввода-вывода со стандартными устройствами, такими, как клавиатура, видеоадаптер, последовательный и параллельный порты ввода-вывода.

Назначение, структура и использование *таблицы векторов прерываний области данных BIOS* рассматриваются в 7-й главе учебного пособия и исследуются при выполнении ряда лабораторных работ.

Таблица 4.1 – Стандартное распределение адресного пространства ПК

64-килобайтовые блоки памяти		Области памяти		
Нач. адрес	Имя	Размер	Имя	Использование
[F0000]h	F	256 Кб	ПЗУ (ROM)	<p>Постоянная память: ПЗУ, ROM – <i>Read Only Memory</i>, доступная только для чтения.</p> <p>Используется для хранения неизменяемых программ и данных: программа самотестирования <i>POST (Power On Self Testing)</i>, автоматически запускаемая при включении питания; программа начальной загрузки MS DOS; компоненты ROM BIOS; знакогенераторы.</p>
[E0000]h	E			
[D0000]h	D			
[C0000]h	C			
[B0000]h	B	128 Кб	Video RAM	<p>Специализированная ОЗУ – буфер обмена данными между программами и видеосистемой.</p> <p>Программа, выполняемая центральным процессором, записывает данные в активную страницу видеопамяти.</p> <p>Программа, выполняемая видеоадаптером, сканирует активную страницу видеопамяти, считывает записанные в нее данные и передает их видеоадаптеру.</p> <p>Видеоадаптер принимает прочитанные данные и вырабатывает сигналы управления дисплеем, формирующие экранный образ активной страницы видеопамяти.</p>
[A0000]h	A			

[90000]h	9	640 Кб	ОЗУ (RAM)	<p>Оперативная память: ОЗУ, RAM – <i>Random Access Memory</i> – память с произвольным доступом, доступная как для чтения, так и для записи.</p> <p>В ОЗУ загружаются системные и пользовательские программы и обрабатываемые ими данные: таблица векторов прерываний, область данных BIOS, данные и программные компоненты MS DOS, драйверы внешних устройств, резидентные программы; пользовательские программы и данные.</p> <p>Детали распределения адресного пространства ОЗУ между системными данными и программами приведены в таблицах 4.2 и 4.3.</p>
[80000]h	8			
[70000]h	7			
[60000]h	6			
[50000]h	5			
[40000]h	4			
[30000]h	3			
[20000]h	2			
[10000]h	1			
[00000]h	0			

Таблица 4.2 – Стандартное распределение оперативной памяти

Нач. адрес	Размер	Назначение области ОЗУ
[0000:0000] _h	1 Кб	<i>Таблица векторов прерываний</i> – справочник начальных адресов программ обработки аппаратных и программных прерываний.
[0040:0000] _h	256 б	<i>Область данных BIOS</i> (детали – в таблице 4.3)
[0050:0000] _h	~ 128 Кб	<p><i>Область MS DOS :</i></p> <ul style="list-style-type: none"> - <i>Программы обработки прерываний</i>, обслуживающие системный ввод/вывод. - <i>Стеки</i> – используются программами обработки прерываний MS DOS. - <i>Переменные окружения</i> операционной системы – задаются командами SET, PATH, PROMPT и др. - <i>Буферы ввода/вывода</i> дисковых накопителей. - <i>Дескрипторы открытых файлов</i>. Каждый дескриптор содержит таблицу доступа к файлу. - <i>Драйверы</i> дополнительных устройств. - Резидентная часть командного процессора (интерпретатора командной строки) <i>COMMAND.COM</i>. - <i>Резидентные программы</i>.
[0050:0080] _h	~512 Кб	Выполняемая прикладная программа

Таблица 4.3 – Структура области данных BIOS

Начальный адрес	Длина, байт	Назначение
[0040:0000] _h	2	Базовый адрес порта первого адаптера последовательного канала (COM1)
[0040:0002] _h	2	То же для COM2
[0040:0004] _h	2	То же для COM3
[0040:0006] _h	2	То же для COM4
[0040:0008] _h	2	Базовый адрес порта для 1-го адаптера параллельного канала (LPT1)
[0040:000A] _h	2	То же для LPT2
[0040:000C] _h	2	То же для LPT3
[0040:000E] _h	2	То же для LPT4
[0040:0010] _h	2	Список установленного оборудования
[0040:0013] _h	2	Общая память (в килобайтах)
[0040:0017] _h	2	Флаги клавиатуры

Продолжение таблицы 4.3

Начальный адрес	Длина, байт	Назначение
[0040:0019] _h	1	Текущее (накопленное) значение ввода кода символа (Alt + цифра)
[0040:001A] _h	2	Адрес «головы» буфера клавиатуры
[0040:001C] _h	2	Адрес «хвоста» буфера клавиатуры
[0040:001E] _h	32	Буфер клавиатуры
[0040:0049] _h	1	Номер текущего видеорежима
[0040:004A] _h	2	Число символов в строке
[0040:004C] _h	2	Размер видеостраницы (в байтах)
[0040:004E] _h	2	Начальный адрес (смещение в видео-сегменте) активной видеостраницы
[0040:0050] _h	8 * 2	Координаты курсора (номера строк и столбцов знаков) для каждой из 8 видеостраниц: младший байт – номер столбца, старший байт – номер строки.
[0040:0060] _h	2	Размер курсора (номера строк пикселей «внутри» знака): младший байт – последняя (верхняя) строка; старший байт – начальная (нижняя) строка.
[0040:0062] _h	1	Номер текущей активной видеостраницы
[0040:0063] _h	4	Адреса регистров видеоконтроллера
[0040:0067] _h	5	Область данных программы POST
[0040:006C] _h	4	Счетчик тиков таймера (текущее время в 55-миллисекундных единицах)
[0040:0080] _h	2	Эффективные адреса (смещения относительно начала сегмента 40 _h) начала и конца буфера клавиатуры. Стандартно – соответственно, [001E] _h и [001C] _h)
[0040:0082] _h	2	
...	...	
[0040:00F0] _h	16	Область для обмена данными между приложениями

4.6.3 Адресное пространство ввода-вывода

Это адресное пространство представлено множеством адресов портов ввода-вывода – однобайтовых регистров контроллеров (адаптеров), через которые производится обмен данными с подключенными к ним периферийными устройствами. Все порты ввода-вывода определенным образом пронумерованы, и номер порта – это и есть его адрес, по которому к порту могут обращаться различные машинные команды.

Существует два основных подхода к организации доступа к портам ввода-вывода: «*Memory mapped I/O*» и «*I/O mapped I/O*».

Первый подход (рисунок 4.8) предполагает отображение портов ввода-вывода на адресное пространство основной памяти.

Общее адресное пространство (1 Мб)	FFFFF _h	Адресное пространство модулей памяти	
	FFFFE _h		
	...		
	BFFFF _h	Адресное пространство ввода-вывода	Port №n

	B0000 _h		.
	AFFFF _h		.
	...		Port №3
	A0000 _h		Port №2
			Port №1
	...	Адресное пространство модулей памяти	
	00001 _h		
	00000 _h		

Рисунок 4.8 – Отображение портов ввода-вывода на адресное пространство модулей памяти (*Memory mapped I/O*)

При этом адресное пространство становится «общим», часть его линейных адресов (на рисунке показаны условно) резервируются для портов ввода-вывода, а оставшиеся адреса используются для обращения к модулям памяти, объем которых при этом, естественно, сокращается.

Преимуществами такого подхода является простота программирования и возможность использования для портов ввода-вывода всего многообразия машинных команд, оперирующих с памятью, а основным недостатком (кроме сокращения основного адресного пространства) – потери производительности за счет увеличения времени ожидания откликов периферийного оборудования.

Второй подход (рисунок 4.9) предполагает автономное использование двух изолированных друг от друга адресных пространств – 1 Мб для модулей памяти и 64 Кб для портов ввода-вывода. В такой ситуации два адресных пространства пересекаются, и адреса портов ввода-вывода совпадают с адресами всех ячеек нулевого сегмента модулей памяти.

При автономном использовании двух адресных пространств и ячейки модулей памяти, и порты ввода-вывода адресуются по общей системной шине, сопряженной с внутренней шиной микропроцессора через два буфера (рисунок

4.3). При этом для адресации ячеек памяти используется все 20 адресных линий [AD0÷AD15] + [A16÷A19], а для адресации портов ввода-вывода – только 16 младших линий [AD0÷AD15].

16-разрядная адресация портов ограничивает объем адресного пространства ввода-вывода величиной в 64 Кб, что позволяет отказаться от сегментирования этого адресного пространства (или условно считать, что все оно расположено в единственном нулевом сегменте). Следует заметить, что 65536 восьмибитных каналов связи с внешним оборудованием – это более, чем достаточно для персонального компьютера.

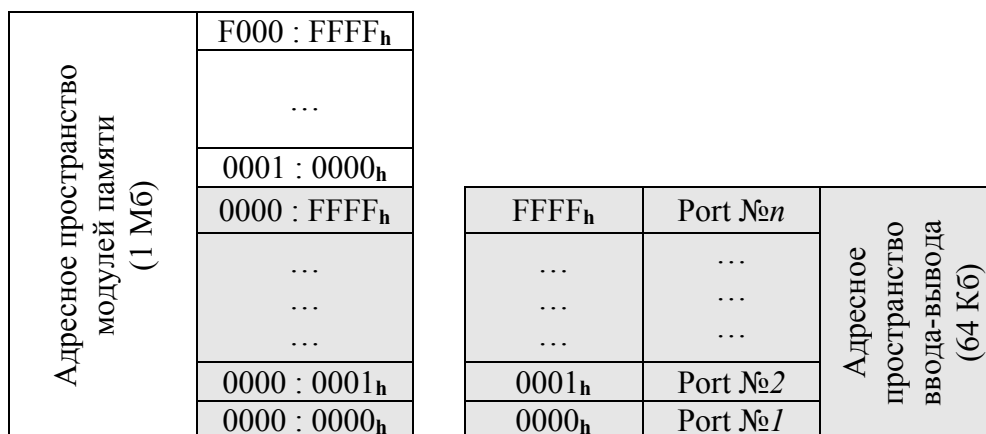


Рисунок 4.9 – Автономное адресное пространство ввода-вывода (I/O mapped I/O)

Проблема пересечения двух адресных пространств в микропроцессорах Intel x86 решена программно-аппаратным способом. В системе команд этого процессора для чтения/записи ячеек модулей памяти используется команда MOV (в ее многочисленных модификациях), а для обмена данными с портами ввода-вывода предусмотрены две специальные одноадресные команды IN и OUT, позволяющие прочитать (IN) данные из порта, адрес которого указан в команде, с их последующей записью в регистр-аккумулятор процессора, или записать (OUT) содержимое регистра-аккумулятора в адресуемый порт.

Дешифратор команд, входящий в состав устройства управления, идентифицирует очередную команду и, в зависимости от записанного в ней кода операции, устанавливает выходную линию M/IO устройства управления (рисунок 4.3) в соответствующее состояние.

Если идентифицирована команда MOV (или другая команда обмена данными с памятью), выходная линия M/IO устройства управления устанавливается в состояние «M», и в результате контроллер системной шины переключает ее в

режим доступа к основному адресному пространству для последующего обмена данными с ячейками модулей памяти.

Если идентифицирована команда IN или OUT, выходная линия M/IO устанавливается в состояние «IO», и контроллер системной шины переключает ее в режим доступа к адресному пространству ввода-вывода, после чего на эту шину из буфера передается заданный в команде адрес порта.

Микропроцессор Intel 8086 позволяет использовать оба режима доступа к адресному пространству ввода-вывода, и эта возможность была реализована в IBM PC и его последующих аналогах для взаимодействия с видеоадаптерами. В этих ПК управляющие регистры видеоадаптера доступны центральному процессору по автономной шине ввода-вывода (*I/O mapped I/O*), а весь объем видеопамяти (128Кб в видеоадаптерах ранних моделей) отображается на основное адресное пространство (*Memory mapped I/O*), занимая в нем диапазон ячеек с адресами от [A000:0000]_h до [B000:FFFF]_h.

16-разрядная адресная шина ввода-вывода обеспечивает возможность доступа к 65536 однобайтовым портам ввода-вывода в диапазоне адресов от [0000]_h до [FFFF]_h, однако в первых IBM-совместимых ПК стандартно использовалось только 10 младших битов этой адресной шины, что уменьшало верхнюю границу адресного пространства до [03FF]_h, а максимальное количество адресуемых портов ввода-вывода – до 1024-х.

Часть адресного пространства ввода-вывода зарезервирована микропроцессором и используется им в своих «внутренних интересах», остальные порты ввода-вывода могут использоваться для подключения адаптеров различных периферийных устройств. В таблице 4.4 приведены примеры закрепления адресов за портами ввода-вывода, обслуживающими клавиатуру, дисковые запоминающие устройства, видеоадаптеры, звуковые карты, устройства, подключаемые через стандартные последовательные и параллельные интерфейсы (принтеры, сканеры и другое периферийное оборудование), а также за контроллером прерываний и контроллером прямого доступа к памяти.

Один контроллер может использовать несколько портов ввода-вывода, назначение которых определяется производителем оборудования или соответствующим стандартом. Например, контроллер *параллельного* интерфейса, к которому обычно подключались матричные принтеры, использует три порта ввода-вывода (регистр *вывода данных*, регистр *состояния* и регистр *управления*), а контроллер *последовательного* интерфейса занимает 7 портов в адресном пространстве ввода-вывода.

Адрес младшего из портов контроллера называют его *базовым адресом*. В процессе начальной загрузки системы базовые адреса контроллеров записываются в *область данных BIOS*, (рисунок 4.7 и таблица 4.3) для оперативного использования программами обмена данными с периферийным оборудованием.

Таблица 4.4 – Примеры использования адресов портов ввода-вывода

Диапазон адресов	Контроллер	Диапазон адресов	Контроллер
0000–000F	Контроллер DMA	0378–037F	Параллельный порт LPT2
0020–0021	Контроллер прерываний	03B0–03BB	Монохромный адаптер (MDA)
0040–0043	Системный таймер	03BC–03BF	Параллельный порт LPT1
0060	Контроллер клавиатуры	03C0–03CF	Видеоадаптер EGA
0070–007F	CMOS-память	03C0–03DF	Видеоадаптер VGA
0278–027F	Параллельный порт LPT3	03D0–03DF	Видеоадаптер CGA
02C0–02DF	Видеоадаптер EGA	03E8–03EF	Последовательный порт COM3
02F8–02FF	Последовательный порт COM2	03F0–03F7	Контроллер гибкого диска
0320–032F	Контроллер жесткого диска	03F8–03FF	Последовательный порт COM1

4.7 Контрольные задания

Задание 4.1: Какой из принципов фон-Неймана определяет структуру ЭВМ и необходимый состав ее функциональных компонентов?

Задание 4.2: Что такое «машинное слово»? Как влияет разрядность АЛУ, разрядность шины данных и адресной шины на производительность работы и на объем памяти компьютера?

Задание 4.3: Определите максимально-допустимый объем адресного пространства в компьютере с 16 (20-, 32-х, 64-х)-разрядной системной адресной шиной.

Задание 4.4: Как влияет емкость установленного ОЗУ на производительность работы компьютера?

Задание 4.5: Как в 16-разрядном микропроцессоре Intel 8086 решена проблема расширения адресного пространства до 1 Мб? Перечислите регистры микропроцессора Intel 8086, участвующие в формировании адреса.

Задание 4.6: Определите понятия «сегмент памяти», «базовый адрес сегмента», «сегментный адрес», «линейный адрес». Какую функцию выполняет сумматор адреса?

- Задание 4.7: Определите 20-разрядные линейные адреса трех однобайтовых ячеек памяти по их сегментным адресам: $[1234:1234]_h$, $[ABCD:ABCD]_h$ и $[ABCD:ABCD]_h$. Предложите несколько альтернативных способов записи сегментных адресов этих ячеек. Оцените общее количество различных способов записи сегментного адреса одной ячейки памяти.
- Задание 4.8: Каково основное назначение регистра флагов и регистров общего назначения (Р.О.Н.) в микропроцессорах x86?
- Задание 4.9: Приведите примеры периферийных устройств персонального компьютера. Почему все они получили такое обобщающее наименование?
- Задание 4.10: Сравните два способа доступа к портам ввода-вывода: «*Memory mapped I/O*» и «*I/O mapped I/O*». Каково назначение линии «M / IO» управляющего устройства микропроцессора Intel 8086?
- Задание 4.11: Какую функцию выполняет контроллер прерываний? Определите термины «сигнал запроса прерывания», «приоритет прерывания», «номер прерывания» и «маскирование прерывания».
- Задание 4.12: Каково назначение регистров *IRR*, *IMR* и *ISR* контроллера прерываний Intel 8259A?
- Задание 4.13: Каково назначение линий *INT* и *INTA* микропроцессора Intel 8086 и линий *INTR* и *INTA* контроллера прерываний?
- Задание 4.14: Проведите анализ устройств HMDD и SSD, представленных на рынке ВЗУ. Сравните самые дорогие и самые дешевые модели этих устройств по основным техническим параметрам. Рассчитайте (приблизительно) стоимость хранения 1 Мб информации для HMDD- и SSD-устройств одинаковой емкости.
- Задание 4.15: Как расшифровывается термин CMOS (КМОП)? Каковы специфические свойства CMOS-памяти? Как используется модуль CMOS-памяти в персональных компьютерах?
- Задание 4.16: Определите модель и основные технические характеристики центрального процессора и видеоадаптера, установленных в Вашем компьютере.

5.1 Классификация программного обеспечения

Вычислительная система – это единый программно-аппаратный комплекс, компоненты которого неразрывно связаны и функционируют в тесном взаимодействии друг с другом. Программное обеспечение (ПО) – необходимый компонент любой компьютерной системы, обеспечивающий ее гибкость, многофункциональность и возможность специализации на решение определенного класса задач в соответствии с потребностями пользователей компьютера.

По назначению и характеру использования различают *системное* и *прикладное* ПО. *Системное ПО* объединяют понятием *операционной системы* (ОС), компоненты которой обеспечивают управление аппаратурой компьютера, хранение данных, управление процессом выполнения программ, взаимодействие с пользователями. ОС современного компьютера – весьма сложный программный комплекс, изучению которого посвящены специальные дисциплины учебных планов подготовки IT-специалистов.

Все остальное ПО относят к категории *прикладного*, подчеркивая этим его вторичность по отношению к системному ПО. Вторичность прикладного ПО (или просто *приложений* – *applications*) по отношению к системному следует понимать в том смысле, что приложения не могут функционировать вне операционной системы: во-первых, процессы их сохранения, загрузки и исполнения управляются операционной системой и, во-вторых, приложения используют различные компоненты системного ПО (например, для обмена данными с периферийными устройствами компьютера).

Компоненты прикладного ПО специализируют компьютер в определенной области и разрабатываются с учетом потребностей определенных категорий пользователей, для которых соответствующее ПО является, по существу, инструментом, используемым в их профессиональной деятельности, в бытовой сфере или в сфере развлечений. Примеры использования прикладного ПО можно взять практически из любой области.

В отдельную категорию выделяют так называемое *инструментальное ПО*, занимающее промежуточное положение между системным и прикладным и ориентированное на особую категорию пользователей – специалистов в области разработки ПО и эксплуатации компьютерных систем (программисты, системные администраторы и др.).

К инструментальному ПО относят программы, осуществляющие управление ресурсами вычислительных систем и ориентированные на решение определенного (как правило – весьма широкого) класса задач: CASE-средства (*Computer Aided Soft Engineering* – компьютерные средства разработки ПО, включающие средства поддержки программных проектов, трансляторы с языков высокого уровня, отладчики программного кода), сетевые сканеры и анализаторы памяти компьютера, серверы баз данных и коммуникационные серверы, средства обеспечения информационной безопасности и т.д.

С одной стороны, инструментальное ПО близко к прикладному, так как не работает на прямую с первичными ресурсами, а использует для этого сервисы, предоставляемые системным ПО. Но с точки зрения технологий разработки и эксплуатации, инструментальное ПО ближе к системному, так как создается для многоцелевого использования, управляет прикладными программами и использует низкоуровневые системные ресурсы в существенно большей степени, чем прикладное ПО.

Далее в этой главе учебного пособия будет рассмотрена функциональная структура MS DOS – одной из первых операционных систем, использовавшихся в IBM-совместимых ПК, и достаточно простой для первоначального ознакомления. Лабораторный практикум (глава 8 учебного пособия) также базируется на виртуальной DOS-машине, а выполнение лабораторных работ потребует освоения и использования специализированного инструментального ПО для исследования алгоритмов функционирования различных компонентов MS DOS, и структур данных, используемых этими алгоритмами.

5.2 Системное ПО

5.2.1 Программная структура MS DOS

Дистрибутив (комплект поставки) операционной системы MS DOS (*MicroSoft Disc Operation System*) включал файлы собственно операционной системы *IO.sys* и *MSDOS.sys*, конфигурационный файл *CONFIG.sys*, командный процессор *COMMAND.com*, программные утилиты, реализующие ряд внешних команд ОС, драйверы периферийных устройств и другие файлы.

Файл IO.sys содержит компоненты базовой системы ввода/вывода BIOS – расширение ROM BIOS, хранимой в ПЗУ компьютера и используемой для взаимодействия с аппаратурой компьютера.

Файл MSDOS.sys – это тело операционной системы, содержит набор программ обработки прерываний, в частности, программу обработки DOS-прерывания INT 21h.

CONFIG.sys – файл текстового формата, включающий набор *команд конфигурации*, которые интерпретируются в процессе загрузки системы и оказывают воздействие на определенные аспекты ее функционирования. Например, команды FCBS и FILES устанавливают максимально-допустимое количество одновременно «открытых» файлов, команда DRIVERS определяет перечень подключаемых драйверов внешних устройств, а командой BUFFERS можно установить количество 512-байтовых буферов (от 2-х до 99), резервируемых в ОЗУ для хранения содержимого секторов системной области дисковых устройств¹⁴ с целью сокращения частоты обращений к внешней памяти при выполнении операций доступа к файлам и каталогам.

Командный процессор COMMAND.com предназначен для организации диалога с пользователем и интерпретации содержимого командной строки¹⁵. Он анализирует вводимые пользователем команды и организует их выполнение, вызывая соответствующие функции DOS или внешние программы (исполнимые файлы форматов .com, .exe или .bat). Поиск программных файлов в файловой системе ПК, их загрузку в ОЗУ и запуск на выполнение также организует командный процессор, используя для этого соответствующие компоненты системного ПО, обслуживающие файловую систему.

Драйверы - это программы, обслуживающие периферийные устройства. Как правило, программы взаимодействуют с устройствами не непосредственно, а через драйверы (рисунок 5.1), поэтому применение драйверов решает проблемы использования в компьютерной системе нового периферийного оборудования – достаточно написать для нового устройства драйвер (если его не поставил разработчик устройства) и подключить его к операционной системе.

¹⁴ Рекомендуется устанавливать количество буферов, как минимум, достаточное для хранения в ОЗУ всей таблицы FAT (п. 6.1), так как именно к этой структуре данных наиболее часто обращаются функции файловой системы.

¹⁵ В MS DOS отсутствовал графический «оконный» интерфейс, привычный современному пользователю ПК, и командный интерфейс был единственным способом общения пользователя с ОС. В ОС семейства Windows, пришедших на смену MS DOS, командный пользовательский интерфейс поддерживается Windows-приложением CMD.exe, которое, по существу, является функциональным аналогом командного процессора COMMAND.com.

Файлы внешних команд операционной системы содержат программные утилиты, предназначенные для выполнения различных «системных» операций (форматирование дисков, дефрагментация дискового пространства, специального копирования файлов и ряд других).

Например, утилита FDISK предназначена для подготовки к работе жесткого диска. Она разбивает диск на участки, называемые *логическими разделами* или *томами*, при этом некоторые разделы объявляются «загружаемыми»¹⁶ (bootable) – с одного из этих разделов (в соответствии с установленными приоритетами) будет производиться загрузка операционной системы. Структура логического раздела диска является объектом исследования при выполнении одной из лабораторных работ, связанных с изучением файловой системы ПК.

Файл AUTOEXEC.bat – файл текстового формата, каждая строка которого содержит DOS-команду. Этот файл (при его наличии) будет автоматически выполнен (то есть будут выполнены все включенные в него команды) в процессе начальной загрузки ОС.

5.2.2 Процесс загрузки MS DOS

MS DOS – это *дисковая* операционная система, представленная множеством программных и конфигурационных файлов, которые хранятся на системном диске, следовательно, процедура ее загрузки при включении питания компьютера должна предусматривать предварительную инициализацию этого диска для последующего считывания с него системных файлов. Проблема в том, что компоненты ОС, обеспечивающие доступ к файловой системе, также хранятся в системных программных файлах и поэтому не могут быть использованы до завершения ее загрузки. Устранение этой проблемы (точнее – ее обход) базируется на двух принципиальных решениях.

Во-первых, не все системные компоненты хранятся в файлах – часть функций ввода-вывода (ROM BIOS) реализуется компонентами, записанными в постоянную энергонезависимую память, там же хранятся и две важные программы, запуск которых не требует участия операционной системы.

Во-вторых, процесс загрузки системы реализуется поэтапно – на каждом этапе, исключая последний, выполняются программы-загрузчики, размещение которых фиксировано и не требует привлечения сервисов файловой системы:

¹⁶ Установка иерархии (приоритетов) активных разделов, с которых будет производиться загрузка ОС, производится соответствующим программированием CMOS-памяти.

каждая программа-загрузчик, выполнив «порученную» ей подготовительную работу, загружает следующую программу-загрузчик, а последняя из них загружает системный файл с программными сервисами файловой системы, с помощью которых затем загружаются и остальные системные файлы.

Процедура загрузки MS DOS при включении питания компьютера реализуется следующими последовательными шагами:

Шаг 1: Управление передается программе *POST (Power On Self Testing)*, которая хранится в ПЗУ по фиксированному адресу (в разделе ROM BIOS). Основное назначение программы – тестирование аппаратных узлов компьютера и их инициализация, предполагающая сохранение параметров конфигурации устройств в соответствующих ячейках *области данных BIOS*.

Программа *POST* при включении питания выполняет следующие действия:

- тестирование *процессора и постоянной памяти*;
- инициализация *таймера и портов ввода-вывода*;
- инициализация *контроллера прямого доступа к памяти*;
- проверка системы *регенерации ОЗУ*;
- тестирование *ОЗУ*;
- загрузка в *ОЗУ части векторов прерываний*;
- инициализация *видеоконтроллера*, после чего все диагностические сообщения будут выводиться на экран;
- инициализация *клавиатуры, CMOS-памяти и системных часов*;
- инициализация *SOM- и LPT-портов*;
- инициализация *контроллеров накопителей* на магнитных дисках.

Шаг 2: Если процесс тестирования и инициализации завершился успешно, запускается хранимая в ПЗУ *программа начальной загрузки (Bootstrap Loader¹⁷)*, которая:

- обращается к *CMOS-памяти* (которая уже инициализирована на первом шаге начальной загрузки, и, следовательно, известен адрес порта *CMOS-памяти*) для чтения таблицы приоритетов загрузочных устройств;

¹⁷ Существует версия происхождения названия *Bootstrap Loader*, которым принято обозначать процесс начальной загрузки операционной системы. Казалось бы, какая связь между ботинками (*boots*) и загрузкой операционной системы? Одно из значений *bootstrap* – «*петля на заднике ботинка, облегчающая его натягивание на ногу*», другое – «*добиваться чего-либо самостоятельно, без посторонней помощи*», есть еще и третье (явно, не последнее) – «*вытащить себя за шнурки собственных ботинок*». Следует согласиться, что у всех перечисленных значений *bootstrap* (особенно – у последнего) просматривается явная аналогия с процессом поэтапной загрузки ОС.

- по таблице приоритетов определяет загрузочный диск (контроллер которого также уже прошел процедуру инициализации);
- обращается к этому диску, загружает из его нулевого сектора *главную загрузочную запись MBR (Master Boot Record)*, в состав которой входит *таблица разделов диска (Partition Table)*¹⁸ и следующая (уже вторая) программа-загрузчик.
- передает управление второму загрузчику.

Шаг 3: Вторая программа-загрузчик:

- читает *таблицу разделов*, содержащую адреса (диапазоны номеров секторов), выделенные разделам диска при его форматировании;
- определяет адрес начального сектора загрузочного раздела диска;
- загружает из этого сектора таблицу параметров форматирования раздела и следующую (третью по порядку) программу-загрузчик;
- передает управление третьему загрузчику.

Шаг 4: Третья программа-загрузчик загружает системный файл IO.sys и MSDOS.sys (они должны быть записаны на диск и доступны через корневой каталог) и передает управление программе IO.sys.

Шаг 5: С системного диска загружается файл CONFIG.sys и описанные в этом файле драйверы.

Шаг 6: С системного диска загружается резидентная часть командного процессора COMMAND.com (содержащего, в частности, ссылки на системные функции – обработчики внутренних команд MS-DOS) и ему передается управление.

Шаг 7: С системного диска загружается файл AUTOEXEC.bat (если он зарегистрирован в корневом каталоге), и выполняются содержащиеся в нем внутренние и/или внешние команды.

Шаг 8: После загрузки командного процессора и выполнения «автосыполняемых» команд, подготовка системы к работе считается завершенной, COMMAND.com выводит на экран так называемое «приглашение DOS» и

¹⁸ Структура главной загрузочной записи, таблицы разделов и таблицы параметров форматирования разделов диска (используемой на следующем – третьем шаге процедуры загрузки ОС) детально рассматриваются в п. 5.5 учебного пособия.

ожидает от пользователя ввода очередной команды для ее последующей интерпретации и исполнения.

5.2.3 Функциональная структура MS DOS

Состав функциональных компонентов ОС и схема взаимодействия системного ПО с аппаратным комплексом и прикладными программами показаны на рисунке 5.1.

Ядро ОС включает несколько подсистем, каждая из которых отвечает за выполнение определенного класса задач. Эти подсистемы общаются с аппаратурой ПК через BIOS, через драйверы или напрямую.

Прикладное ПО может обращаться к драйверам через соответствующую подсистему ОС, работать с BIOS или непосредственно с аппаратурой. Очевидно, что чем выше уровень интерфейса прикладной программы и аппаратуры, тем меньше программа будет зависеть от особенностей аппаратуры.

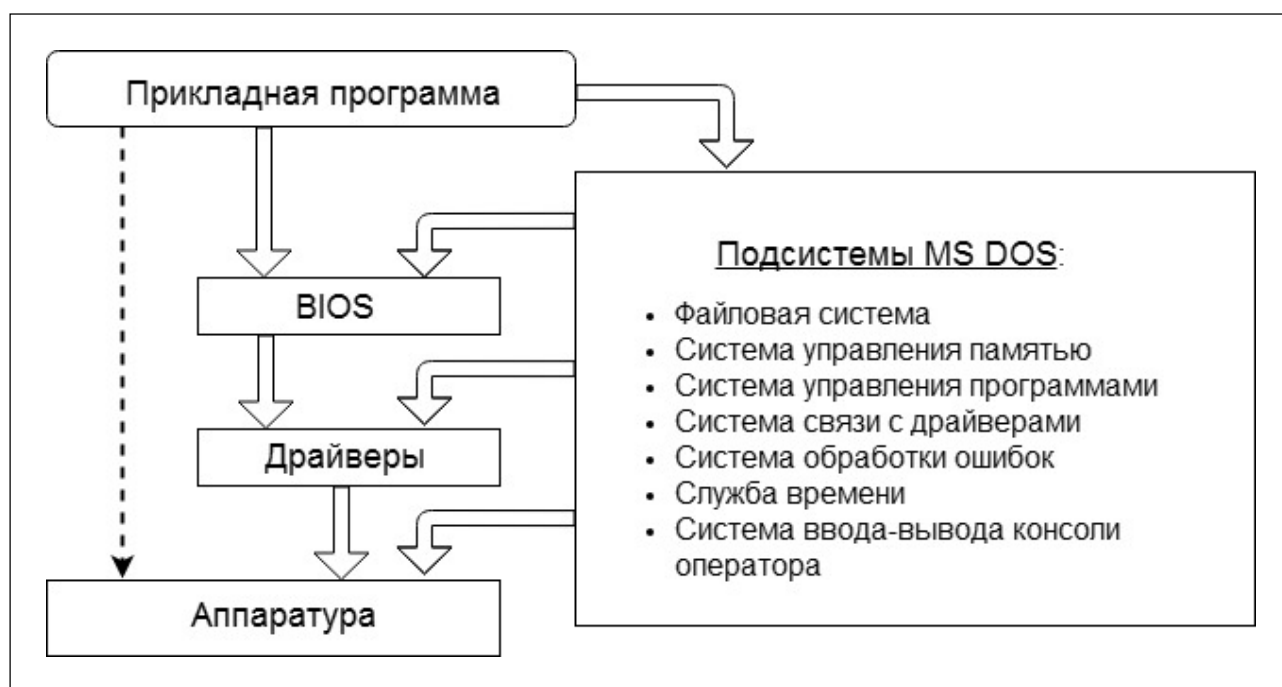


Рисунок 5.1 - Схема взаимодействия ПО с аппаратурой ПК

Файловая система является одной из важнейших подсистем MS DOS, она используется как в процессе загрузки ОС, так и в процессе ее работы. Сама операционная система хранится файловом формате, все прикладные программы и обрабатываемые ими данные хранятся в виде файлов, поэтому можно смело утверждать, что файловая система – ключевая подсистема ОС. Файловая система включает множество программ, доступных через функции соответствующих

программных прерываний, а также используемые этими программами структуры данных, организованные на устройствах внешней памяти.

Файловая система работает с дисками через драйверы, а драйверы, в свою очередь, пользуются сервисами BIOS. На уровне BIOS выполняются элементарные дисковые операции, такие, например, как чтение/запись секторов, форматирование диска и др. Такой низкоуровневый доступ к диску возможен и непосредственно из прикладной программы, но обычно она пользуется соответствующими функциями прерывания DOS. Программы защиты от несанкционированного доступа или копирования файлов вынуждены обращаться к средствам более низкого уровня, вызывая прерывания BIOS, или даже работать с контроллером дисковода через порты ввода/вывода.

Рассмотрению двух популярных файловых систем посвящен следующий раздел учебного пособия, в одной из лабораторных исследуются алгоритмы выполнения файловых операций, реализованных в FAT-ориентированных файловых системах ПК.

Система управления памятью используется для выделения памяти, необходимой выполняемым программам. Вся доступная программам память разбивается на блоки, каждому блоку предшествует управляющая структура MCB (*Memory Control Block*), в которой записаны характеристики блока. Все блоки располагаются друг за другом, адрес первого блока хранится в специальной структуре данных – организуемой в ОЗУ векторной таблице связи *CVT* – *Communications Vector Table*.

Для каждой вновь запускаемой прикладной программы MS DOS создает определенное количество MCB-блоков. Программа может «заказать» для себя дополнительные блоки памяти, используя соответствующие функции DOS-прерывания 21_h, обеспечивающие доступ к системе управления памятью. При освобождении памяти или при поступлении от программ запросов на получение дополнительной памяти, содержимое таблицы *CVT* соответственно корректируется.

Система управления программами при запуске прикладной программы последовательно выполняет типовой набор операций:

- обращается к *системе управления памятью* для подготовки блоков памяти для запускаемой программы;
- обращается к *файловой системе* и загружает в ОЗУ файл (или фрагмента файла), содержащий программу;
- передает управление программе.

Другая задача, решаемая системой управления программами – это запуск программ из других программ и организация программных перекрытий (overlay), используемых в условиях недостаточного объема оперативной памяти. Если не все компоненты большого программного комплекса нужны одновременно, можно разбить комплекс на несколько оверлейных модулей, загружаемых последовательно в один и тот же блок памяти.

Еще одна функция системы управления программами связана с организацией работы *резидентных программ*. Если после завершения работы программы необходимо оставить её в памяти для последующих оперативных запусков (то есть сделать эту программу резидентной), следует обратиться к системе управления программами через соответствующую функцию DOS-прерывания 21h. Классическими примерами резидентных программ являются интерпретатор команд COMMAND.com и анализатор памяти Peek-Pook-Resident (файл Peek.com), который будет использоваться в качестве основного инструмента при выполнении ряда лабораторных работ.

Система связи с драйверами периферийных устройств обеспечивает интерфейс между программным обеспечением и аппаратурой компьютера и существенно продлевает срок жизни прикладного и системного ПО.

На рынке регулярно появляется новое периферийное оборудование и новые модификации уже существующих устройств, и это объективная реальность, бороться с которой бессмысленно и бесперспективно. Но что делать с уже разработанным ПО, рассчитанным на старую аппаратуру? Если ПО еще не устарело, вряд ли целесообразно переделывать его только из-за того, что появилось устройство, поддерживающее новый протокол обмена. Интуитивно понятно, что в этих условиях должна существовать какая-то программная прослойка между аппаратным и программным обеспечением, выполняющая согласующие функции и избавляющая прикладные (а в ряде случаев – и системные) программы от необходимости знать технические подробности устройства периферийного оборудования.

Проблемы устранения зависимости программного обеспечения от аппаратуры решаются по-разному в различных операционных системах. MS-DOS, как и другие операционные системы, использует для этих целей механизм драйверов, однако драйверы MS-DOS не всегда обращаются напрямую к аппаратуре, а вызывают функции BIOS, и уже BIOS выполняет операции ввода/вывода.

При этом BIOS обслуживает только стандартные устройства, нестандартные же устройства обслуживаются драйверами напрямую. Использование BIOS в качестве интерфейса между драйверами стандартных устройств и аппаратурой существенно повышает живучесть MS-DOS на ПК, не вполне совместимых с классическими IBM PC. Для управления состоянием устройства ввода/вывода и обмена информацией между прикладной программой и драйвером устройства используется специальная функция 44₁₆ прерывания DOS 21₁₆.

Система обработки ошибок MS DOS использует флаг переноса (CARRY FLAG, CF) регистра флагов центрального процессора. Если после обращения к прерыванию CF установлен в 1, произошла ошибка. Интерпретацию ошибки выполняет соответствующая функция прерывания DOS. Например, если произошла критическая ошибка ввода/вывода, вызывается стандартная процедура DOS, выводящая на экран запрос о дальнейших действиях.

Служба времени.

Компьютер оборудуется системными часами с питанием от аккумулятора, содержимое которых не сбрасывается при выключении питания, и системным таймером, регулярно (каждые 55 мс) вырабатывающим соответствующее прерывание. MS DOS содержит драйвер устройства CLOCK\$, который постоянно ведет подсчет времени и хранит текущие время и дату в соответствующем буфере, доступном прикладным программам через одну из функций прерывания 21h. Прикладная программа может также перехватывать прерывания системного таймера (INT 8_h) и использовать эти прерывания для регулярного выполнения каких-либо функций.

Ввод/вывод на консоль оператора.

Консоль оператора состоит из двух устройств – клавиатуры и дисплея, которые обслуживаются одним драйвером консоли CON. Операционная система обслуживает консоль с помощью функций прерывания 21h, обеспечивающих ввод и вывод символов на устройство CON. Для работы с клавиатурой и дисплейным адаптером на низком уровне этот драйвер использует соответствующие прерывания BIOS.

В качестве практического примера использования консоли оператора можно рассмотреть применение команды копирования файлов COPY: команда *COPY CON <путь к файлу>* запишет в указанный файл текстовые данные, введенные пользователем с клавиатуры, а команда *COPY <путь к файлу> CON* прочитает содержимое указанного в команде текстового файла и выведен его на экран видеомонитора.

5.3 Контрольные задания

Задание 5.1: Определите следующие понятия: *программное обеспечение* (ПО), *системное ПО*, *прикладное ПО*, *инструментальное ПО*. Приведите соответствующие примеры.

Задание 5.2: К какой из перечисленных выше категорий относятся и для чего могут быть использованы следующие программные продукты:

- Adode Reader;
- IDLE;
- HeliconFilter;
- MS Word;
- MS SQL Server;
- OpenOffice Calc;
- StarUML;
- Zoom.

Задание 5.3: К какой категории ПО относятся CASE-средства? Какие их перечисленных выше программных продуктов следует отнести к CASE-средствам?

Задание 5.4: Перечислите основные подсистемы MS DOS и дайте каждой из них краткую характеристику.

Задание 5.5: Какие программы называют *драйверами*? К какой категории ПО следует отнести такие программы?

Задание 5.6: Какие функции выполняют файловые системы? Какие файловые системы используются на внешних ЗУ Вашего компьютера?

Задание 5.7: Какое место занимает в файловой системе MS DOS программа COMMAND.com ?

Задание 5.8: Используя электронный справочник HELP, определите состав функций 21-го прерывания DOS, имеющих отношение к файловой системе.

Задание 5.9: Перечислите компоненты системного ПО, участвующие в процессе начальной загрузки MS DOS. На каких запоминающих устройствах хранятся эти компоненты.

Задание 5.10: Опишите последовательность шагов начальной загрузки MS DOS при включении питания компьютера.

Задание 5.11: Какие функции выполняет программа POST ?

Файловая система является важнейшим компонентом операционной системы компьютера, ее основное назначение – обеспечение надежного хранения файлов на внешних запоминающих устройствах и предоставление прикладным программам унифицированных и эффективных средств доступа к файлам, наличие которых избавляет прикладную программу от необходимости знать технические подробности устройства дисковой системы компьютера.

Файловая система включает программные компоненты, обеспечивающие реализацию типовых операций доступа к файлам, и специальные структуры данных (организуемые как в оперативной, так и в дисковой памяти), обслуживающие работу соответствующих программных компонентов.

6.1 Трехуровневая модель дискового пространства

Файловые системы поддерживают трехуровневую модель памяти, обеспечивающую интерфейс прикладных программ с аппаратурой ВЗУ.

Верхний (пользовательский) уровень доступен прикладным программам, которые отправляют файловой системе запросы на выполнение операций доступа к файлам и получают от нее результаты выполнения таких запросов.

Дисковое пространство ПК на пользовательском уровне представляется (и визуально поддерживается) множеством иерархических (древовидных) структур: в каждом из «деревьев» роль «корня» выполняет *том* (логический диск), ассоциируемый с корневым каталогом тома, роли «ветвей» – *подчиненные* (дочерние) *каталоги* различных уровней подчиненности, а *файлам* в этой «древесной» терминологии отведена роль «листьев», которые всегда являются дочерними объектами, то есть, привязаны к какому-либо (единственному) родительскому каталогу, и при этом сами не могут иметь дочерних объектов.

Программы «общаются» с файловой системой на языке пользовательского представления и оперирует привычными пользователю терминами для обозначения объектов: диск (или том), папка (или каталог), файл определенного типа (приложение, документ), каждый из которых должен быть поименован в соответствии с установленными соглашениями об именовании объектов. Базовым понятием такого представления является «*путь к файлу*» (или к каталогу), который представляется текстовой строкой стандартного формата, практически одинакового в различных файловых системах. В начале строки указывается имя тома, а далее – имена каталогов в порядке их подчиненности.

Если прикладной программе требуется получить доступ к файлу с целью его последующей обработки, программист должен предусмотреть в исходном коде этой программы оператор открытия файла, например (в операционных системах DOS/Windows), такой: *open file C:\my_files\documents\doc123.txt*.

При этом прикладной программист не должен заботиться о реализации операции доступа к файлу и не обязан знать¹⁹, что указанный в команде путь к файлу будет передан для обработки соответствующим программным компонентам файловой системы, которые преобразуют его в аппаратный адрес файла и передадут этот адрес для исполнения контроллеру диска, обеспечивающему чтение фрагмента файла с диска и запись его в буфер ОЗУ для последующей обработки программой.

Пользовательский интерфейс операционной системы предоставляет возможность выполнения типового набора операций над объектами файловой системы, например:

- *диск* можно «открыть», просмотреть его свойства (например, тип файловой системы или объем свободного пространства) или отформатировать, в результате чего все ранее созданные на нем папки и файлы куда-то безвозвратно (как думает пользователь) исчезнут;
- *папку* можно переименовать, открыть и просмотреть ее содержимое, удалить или создать новую папку, подчиненную созданной ранее;
- *файл* можно переименовать, удалить, скопировать или переместить в другую папку, выполнить (если этот файл – приложение) или обработать соответствующим приложением, например: поместить в архив (сжать), отредактировать (если этот файл – документ), просмотреть графический файл, выполнить вычисления в электронной таблице или запрос в базе данных.

На *нижнем (аппаратном) уровне* модели программные компоненты файловой системы «общаются» (обмениваются данными) с контроллерами запоминающих устройств. Язык такого общения – это, по существу, описание устройства дискового накопителя с позиций управляющего им контроллера.

Контроллер диска имеет дело с физической структурой дискового пространства, которая описывается такими терминами, как *диск* (или *пакет дисков*), понимаемый как физическое устройство с автономным приводом, *рабочая поверхность* диска (*Side*), связанная с магнитной головкой чтения-записи

¹⁹ Конечно же, профессиональному прикладному программисту непозволительно не знать основ организации файловых систем.

(*Head*), магнитная *дорожка* (*Track*) и *сектор* (*Sector*). Контроллер управляет приводом вращения пакета дисков, приводом возвратно-поступательного перемещения блока магнитных головок с высокоточной системой позиционирования на заданном радиусе диска, а также процессом последовательного чтения-записи секторов диска магнитными головками²⁰.

Все рабочие поверхности (головки), дорожки (цилиндры) и секторы последовательно пронумерованы, так что номер головки, номер цилиндра и номер сектора вместе однозначно определяют *аппаратный адрес* сектора (называемый *относительным адресом*). Используют также и *абсолютную* нумерацию секторов – начиная с нулевого сектора нулевой дорожки нулевой поверхности диска. Если для диска известно количество головок, цилиндров и секторов на каждой дорожке, и определен порядок их нумерации, относительные адреса секторов легко пересчитываются в их абсолютные адреса и обратно.

Для однозначного определения места расположения файла на диске необходимо связать с этим файлом упорядоченную последовательность пронумерованных секторов.

Промежуточный уровень модели обеспечивает интерфейсы взаимодействия с пользовательским и аппаратным уровнями. На этом уровне поддерживается *ленточная* модель дискового пространства, представляющая его в форме условной ленты, на которой последовательно размещены секторы в порядке их абсолютной нумерации.

Нулевой (в абсолютной нумерации) сектор содержит *главную загрузочную запись* (*Master Boot Record*, MBR), содержащая *исполнимый код программы начальной загрузки ОС* (п. 5.2.2) и *таблицу разделов диска* (*Partition Table*), в которой каждый том представлен 16-байтовым элементом (таблица 6.1).

Сектор – это весьма малая единица емкости дискового накопителя, и использование номера сектора в качестве логического адреса файла снижает эффективность использования дисков большой емкости. Поэтому для адресации файлов на *промежуточном уровне* модели дискового пространства используется другая, более крупная единица, называемая *кластером*.

²⁰ Накопитель при этом может быть совсем не *дисковым* и не *магнитным*, а, например, твердотельным (SSD), в котором трудно отыскать *магнитные головки, дорожки и цилиндры*, но все эти технические детали надежно скрываются от файловой системы контроллером накопителя, который может найти *сектор*, номер которого ему передан, прочесть его содержимое или записать в этот сектор переданную контроллеру порцию данных.

Кластер – это блок из нескольких «соседних» секторов (в смысле их абсолютной адресации), размер кластера задается при форматировании тома. Все кластеры тома последовательно пронумерованы, при этом номер кластера однозначно определяет номера всех входящих в него секторов.

Таблица 6.1 - Структура таблицы разделов диска

Смещение, байт	Длина, байт	Содержимое
1-й раздел диска		
+00	1	Флаг загрузки : 0 – не загружаемый, 80h – загружаемый (<i>Bootable</i>)
+01	1	Начало раздела: HdS (№ головки)
+02	2	Начало раздела: Sec (№ сектора – 6 младших битов) Cyl (№ цилиндра – 10 старших битов)
+04	1	Код системы: 1 – DOS (FAT-12); 4 – DOS (FAT-16)
+05	1	Конец раздела: HdE (№ головки)
+06	2	Конец раздела: Sec и Cyl (аналогично началу раздела)
+08	4	Абсолютный номер начального сектора раздела (соответствует номерам сектора, головки и цилиндра начала раздела) : $Cyl * сект./дор. * дор./цил. + HdS * сект./дор. + (Sec - 1)$
+0Ch	4	Общее количество секторов раздела
2-й раздел диска		
...
Последний раздел диска		
...

Модели логических разделов диска реализуются по-разному в различных файловых системах. Далее будут рассмотрены две таких реализации: система типа FAT, разработанная первоначально для обслуживания накопителей на гибких магнитных дисках малой емкости, и более современная система NTFS.

Одна из лабораторных работ предусматривает проведение экспериментального исследования алгоритмов выполнения типовых файловых операций, реализованных соответствующими программными компонентами FAT-систем.

6.2 Файловые FAT-системы

Как уже отмечалось, жесткий диск может быть разделен на несколько логических разделов (томов), каждый из которых представляется пользователю, как автономный диск. Каждый том занимает определенный ему при форматировании диапазон секторов диска, номера которых указаны в соответствующей строке таблицы разделов. При рассмотрении структуры тома будем использовать внутреннюю нумерацию секторов в пределах тома: первый из выделенных тому секторов, будем называть *нулевым* сектором, следующий – *первым* и т.д.

FAT-том разделен на две расположенные последовательно области – системную и рабочую.

Рабочая область занимает основную часть тома и расположена непосредственно после системной области. Рабочая область (в отличие от системной) разделена на последовательно пронумерованные *кластеры* и предназначена для хранения файлов и подчиненных каталогов (которые, по существу, – тоже файлы специального формата).

Кластер используется в качестве минимальной единицы, выделяемой операционной системой одному файлу, и этот файл будет единственным «владельцем» каждого из выделенных ему кластеров, независимо от того, осталось ли в кластере незанятое пространство после записи в него фрагмента файла.

Каждый кластер имеет уникальный номер и содержит несколько расположенных подряд секторов (как правило, от 2-х до 16, но допускаются и кластеры, состоящие из одного сектора). Номера кластеров используются в качестве адресов расположения файлов, между номером кластера и списком номеров входящих в него секторов существует взаимно-однозначное соответствие.

Системная область (существенно меньшая по размеру) занимает на томе несколько начальных секторов, начиная с нулевого, и используется для хранения служебных структур данных, необходимых для организации доступа к файлам и загрузки операционной системы.

Секторы системной области не объединяются в кластеры, запись файлов любого типа в эту область запрещена.

6.2.1 Структура системной области тома

Системная область содержит три служебные структуры данных:

- *блок загрузки (Boot-Record)*;
- *таблица распределения файлов (File Allocation Table – FAT)*;
- *корневой каталог (Root Directory)*.

6.2.1.1 Блок загрузки

Блок загрузки (таблица 6.2) занимает нулевой сектор и содержит таблицу параметров формата диска и короткую программу загрузки ОС (п. 5.2.2). Все данные блока загрузки записываются на том в процессе его форматирования соответствующей программой, информация о которой сохраняется в первой записи таблицы параметров (8-байтовый текстовый блок во второй строке таблицы 6.2).

Таблица 6.2 – Структура блока загрузки (boot record)

Смещение, байт	Длина, байт	Содержимое элемента блока загрузки
+00	3	Команда перехода на программу загрузки (JMP)
+03	8	Служебная информация программы форматирования
+0B _h	2	Размер сектора в байтах
+0D _h	1	Размер кластера в секторах
+0E _h	2	Количество секторов перед первой таблицей FAT
+10 _h	1	Количество таблиц FAT
+11 _h	2	Максимальное число элементов корневого каталога
+13 _h	2	Общее количество секторов на томе
+15 _h	1	Дескриптор носителя (то же, что 1-й байт FAT)
+16 _h	2	Количество секторов, выделенных одной FAT
+18 _h	2	Количество секторов на дорожке
+1A _h	2	Количество головок чтения/записи (поверхностей)
+1C _h	2	Количество спрятанных секторов
+1EH		Начало программы начальной загрузки ОС

6.2.1.2 Таблица расположения файлов

Таблица расположения файлов (FAT) занимает в системной области фиксированный набор секторов (таблица 6.2), расположенных непосредственно после загрузочного сектора, и используется для хранения номеров свободных кластеров и номеров кластеров, выделенных файлам при их записи.

FAT – это информационная модель рабочей области тома, представленная в форме *списка*, каждый элемент которого содержит числовой (двоичный) код, представляющий некоторую характеристику кластера рабочей области, соответствующего этому элементу (таблица 6.3).

Если при форматировании тома в его рабочей области создано n кластеров, то в системной области будет сформирована структура FAT (точнее – две идентичных копии этой структуры), содержащая список из n элементов, при этом каждый i -тый элемент этого списка FAT[i] представляет соответствующий кластер – i -тый элемент списка CLUST[i].

Размер элемента FAT (разрядность соответствующего двоичного кода) ограничивает количество кластеров, создаваемых в рабочей области тома при его форматировании. Если, например, попытаться отформатировать том в 12-битной системе FAT-12 (когда-то применявшейся для гибких дисков), то на этом томе нельзя будет сформировать более $2^{12} = 4096$ кластеров, а фактически – чуть

меньше, так как 18 кодов зарезервированы для хранения служебной информации и не могут использоваться для «нумерации» кластеров, (таблица 6.3).

Если при этом емкость тома не превышает 2-х Мб (2^{21} б), то его удастся отформатировать с минимально-возможным размером кластера, равным одному сектору ($2^{21} / 2^{12} = 2^9 = 512$ б), но для томов большей емкости, форматируемых в FAT-12, создание таких «мелких» кластеров будет технически невозможным.

Таблица 6.3 – Числовые коды элементов FAT

Значение FAT[i]	Состояние кластера CLUST[i]	Комментарии
(0)000 _h	Кластер свободен и доступен для записи	При форматировании тома (в любом из режимов – <i>quick</i> или <i>full</i>) все кластеры, кроме «сбойных», объявляются свободными, то есть все элементы FAT получают нулевые значения.
от (0)002 _h до (F)FEF _h	Кластер занят и содержит не последний фрагмент файла	Код FAT[i] – это номер следующего кластера, в котором продолжается (или заканчивается) файл, очередной фрагмент которого расположен в <i>i</i> -том кластере.
от (F)FF8 _h до (F)FFF _h	Кластер занят и содержит последний фрагмент файла	FAT[i] – это код окончания «цепочки» кластеров, занятых файлом (<i>EOF – End Of File</i>).
от (F)FF0 _h до (F)FF7 _h	Кластер «сбойный», не доступен для чтения и записи	При <i>полном</i> форматировании тома (<i>full format</i>) весь кластер объявляется «сбойным» (<i>Bad</i>), если «сбойным» оказывается хотя бы один его сектор.

На томе, отформатированном в системе FAT-16, станет возможным создание 65 518 кластеров, а в системе FAT-32 – уже 4 294 967 278, что практически снимает ограничение на минимальный размер кластера: например, для тома емкостью 2 Тб ($\sim 2^{41}$ б), форматируемого в FAT-32, минимальный размер кластера составит один сектор ($2^{41} / 2^{32} = 2^9 = 512$ б) – так же, как и для тома емкостью 2 Мб, отформатированного в системе FAT-12.

Какой же размер кластера следует выбрать при форматировании тома? Всё зависит от его предназначения. Если, например, том является хранилищем коротких сообщений на почтовом сервере, то эффективность использования дискового пространства будет выше при кластерах минимального размера, так как даже самому короткому файлу не может быть выделено меньше одного кластера. Если же том используется на сервере баз данных, файлы которых, как правило, имеют гигабайтные размеры, эффективнее будут крупные кластеры, так как при этом ускоряется обработка FAT и сокращается размер системной области тома.

Программы форматирования предлагают (по умолчанию) размеры кластеров от 4-х до 8 секторов (соответственно, от 2-х до 4-х Кб) в зависимости от емкости форматируемых дисков.

Какую же разрядность FAT (12, 16 или 32 бита) выбрать при форматировании тома? Повышение разрядности экспоненциально увеличивает допустимое количество кластеров, но при этом увеличивается и размер системной области, которая «отнимает» часть дискового пространства у рабочей области тома, используемой для хранения файлов. При этом в ряде случаев, особенно при форматировании томов небольшой емкости, увеличение разрядности FAT бывает избыточным и создает негативный эффект.

Размер фрагмента системной области тома, резервируемого для хранения FAT, определяется количеством ее элементов (равным количеству кластеров рабочей области), и размером каждого такого элемента.

В первом из рассмотренных выше примеров том размером 2 Мб был отформатирован в системе FAT-12 с 512-байтовыми кластерами. При этом в рабочей области было создано приблизительно 4096 кластеров ($2^{21} / 2^9 = 2^{12}$), и столько же 12-битных элементов было сформировано в FAT. В этих условиях каждая из двух копий FAT будет занимать в системной области тома по 12 секторов ($4096 \times 12 / 8 / 512$), отнимая этот объем у рабочей области тома.

Если этот же том отформатировать в системе FAT-16 с таким же размером кластеров, то FAT увеличится до 16 секторов ($4096 \times 16 / 8 / 512$), а при форматировании в FAT-32 – до 32-х секторов ($4096 \times 32 / 8 / 512$).

Как уже отмечалось, структура FAT первоначально разрабатывалась для дисковых устройств небольшой емкости, когда главным критерием качества была эффективность использования дискового пространства при хранении файлов. Принятая в FAT система кодирования кластеров (таблица 6.3), безусловно, соответствует этому критерию, но она противоречит другому, не менее важному, критерию, так как способствует фрагментарному расположению файлов, что увеличивает время доступа к секторам диска на аппаратном уровне.

Для иллюстрации основных концепций FAT рассмотрим небольшой ее фрагмент, описывающий диапазон рабочей области тома из 32-х кластеров – с 10-го по 41-й (рисунок 6.1). На рисунке *курсивом* обозначены номера элементов FAT, а **жирным шрифтом** – числовые коды, записанные в этих элементах. Разумеется, все числовые коды FAT записаны в двоичной системе счисления, но

для удобства прочтения все номера кластеров представлены на рисунке в десятичной системе, а специальные коды EOF и BAD – в шестнадцатеричной.

10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	0	13	14	15	FFF	0	0	19	20	21	31	0	0	0	0
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
0	0	0	0	0	32	33	35	FF7	36	37	FFF	0	0	0	0

Рисунок 6.1 – Фрагмент таблицы FAT

Как следует из таблицы 6.3, кластеры с номерами 10, 11, 16, 17, с 22-го по 30-й и с 38-го по 41-й свободны и доступны для записи, так как в соответствующих этим кластерам элементах FAT записан код «0».

Кластер №33 объявлен «сбойным» (bad-cluster), так как код **FF7_h**, записанный в 33-м элементе FAT, блокирует возможность доступа к этому кластеру программным компонентам FAT. Разумеется, программа форматирования диска такой доступ получит беспрепятственно, как, впрочем, и любая другая программа, работающая с контроллером диска напрямую без использования соответствующих драйверов и функций DOS или BIOS (как это показано пунктирной линией на рисунке 5.1).

В кластерах №15 и №37 заканчиваются какие-то два файла, на что указывает код **FFF** (*EOF – End Of File*) в соответствующих элементах FAT.

Цепочка ссылок «12–13–14–15» в элементах FAT позволяет предположить, что в кластерах с этими номерами последовательно записан какой-то файл, размеры которого не превышают размера 4-х кластеров тома.

Второй файл, заканчивающийся в 37-м кластере, записан на диск фрагментарно и занимает три фрагмента кластеров: [18...21]+[31...33]+[35...37]²¹. Последние два фрагмента разделены BAD-кластером №34, а первые два разделены девятью свободными кластерами – возможно, в момент записи второго файла эти кластеры были заняты другими файлами, которые позднее были удалены, что и привело к «обнулению» соответствующих элементов FAT.

²¹ Алгоритмы поиска свободного дискового пространства, реализуемые функциями FAT при записи файла на диск рассматриваются в п. 6.2.2.4 – они неоднократно изменялись при обновлении версий ОС. Экспериментальному исследованию этих алгоритмов посвящена одна из лабораторных работ, а их программной реализации – одно из контрольных заданий.

Вероятно, по этой же причине между 1-м и 2-м файлами присутствуют свободные кластеры №16 и №17, а также свободные кластеры №10 и №11, почему-то не выделенные первому файлу при его записи на диск.

Структура FAT позволяет эффективно отследить всю цепочку занятых файлом кластеров, при условии, если будет известно начало этой цепочки. Конечно, начальные элементы всех цепочек кластеров можно определить алгоритмически (как элементы, ссылающиеся на другие элементы, но на которые не ссылается ни один из других элементов), но разработчиками FAT было принято другое, более простое решение: номер начального кластера для каждого файла хранится не в FAT, а в специальной структуре данных, называемой *каталогом* (корневым или любым из подчиненных), наряду с другими свойствами файла.

Именно с просмотра каталога и начинается свою работу алгоритм поиска файла, рассматриваемый ниже в п. 6.2.2.2.

6.2.1.3 Корневой каталог

Корневой каталог – это последняя из трех структур данных, создаваемых в системной области тома при его форматировании. Корневой каталог ограничен в размере соответствующим параметром, хранимым в блоке загрузки (таблица 6.2) и занимает последовательность секторов, расположенных непосредственно после второй копии FAT.

Корневой каталог состоит из множества регистрационных записей, каждая из которых содержит информацию об одном файле (в том числе – о файле типа «подчиненный каталог»), зарегистрированном в корневом каталоге.

Запись корневого каталога имеет фиксированную длину в 32 байта, ее структура представлена в таблице 6.4.

Таблица 6.4 – Структура записи корневого каталога

Смещение	Длина, байт	Содержимое
+00	8	Имя файла
+08	3	Расширение файла
+0B _h	1	Атрибуты файла
+0C _h	0A _h	Резерв
+16 _h	2	Время создания/модификации (в спец. формате)
+18 _h	2	Дата создания/модификации (в спец. формате)
+1A _h	2	Номер начального кластера
+1C _h	4	Размер файла в байтах
+20 _h	1	Размер элемента каталога (как правило, равно 20 _h)

Имя и расширение имени файла. Как показано в таблице 6.4, под имя и расширение имени файла отведены два первых поля записи суммарной длиной всего 11 байтов, что соответствует 8-битному ASCII-кодированию текстовых символов и соглашению об именовании файлов, принятому в MS DOS (не более 8 знаков в имени файла и не более трех – в расширении).

Уже в ранних версиях ОС Windows это ограничение было снято, и сегодня полное имя Windows-файла формально может содержать до 255 знаков (с возможностью использования почти всех символов, в том числе нескольких символов «точка», а также символов дополнительных кодовых таблиц), что 8-кратно превышает размер всей регистрационной записи каталога.

Хранение «длинных» имен файлов в «коротких» записях каталогов необходимо для обеспечения совместимости файловых систем MS DOS и Windows, и эта проблема была решена следующим образом:

- из «длинного» Windows-имени по специальному алгоритму выделяются 11 символов (8 + 3), соответствующих требованиям MS DOS;
- это программно сформированное «короткое» имя вместе с остальными параметрами файла хранится стандартным для MS DOS способом в 32-байтовой регистрационной записи каталога;
- исходное «длинное» Windows-имя файла хранится как неструктурированный текст, занимая необходимое количество дополнительных 32-байтовых записей этого же каталога.

Короткое DOS-имя файла формируется из длинного Windows-имени по следующему алгоритму, приводимому с некоторыми упрощениями:

- из исходного длинного имени файла удаляются все запрещенные правилами MS DOS символы, в том числе и пробелы, а также все точки, кроме одной (самой правой) точки, которая получает статус разделителя между именем файла и расширением имени;
- из числа оставшихся символов, расположенных правее точки, отбрасываются все крайние правые символы, кроме трех прижатых к точке символов, которые становятся расширением имени файла;
- из числа символов, расположенных левее точки, оставляются только 6 крайних левых символов, остальные отбрасываются, после чего справа добавляется седьмой символ тильда (~) и восьмой цифровой символ (1 ... 9) в зависимости от количества одноименных файлов, уже зарегистрированных в каталоге (то есть файлов с одинаковыми шестью первыми символами в именах и тремя одинаковыми символами в расширениях);

– если после такого «усечения» имени в нем остались строчные буквы дополнительной кодовой таблицы (например, символы кириллицы), все они заменяются соответствующими прописными буквами (п.6.2.2.5).

Если, например, файл имеет имя «Длинное . Имя . Файла . русское», то будет сформировано его короткое имя «ДЛИННО~1.РУС» или «ДЛИННО~2.РУС», если в каталоге уже имеется один файл с такими же именем и расширением.

Атрибуты файла.

Третье поле записи длиной в 1 байт отведено для хранения *атрибутов* файла – шести битовых флагов, единичное значение каждого из которых задает определенное свойство файла:

0 -й бит = 1 - файл доступен только для чтения (**R/O** – Read Only)

1 -й бит = 1 - файл объявляется *спрятанным* (**Hid** – Hidden), то есть штатные программы просмотра каталогов не должны его показывать в списке файлов

2 -й бит = 1 - системный файл (**Sys** - System)

3 -й бит = 1 - фиктивный файл, имя и расширение которого используются в качестве *метки тома* (**Vol** - Volume)

4 -й бит = 1 - файл специального формата – подчиненный каталог (**Dir** - Directory)

5 -й бит = 1 - файл, требующий архивирования при создании очередной резервной копии диска (**Arc**)

Информация, хранимая в атрибутах файлов, используется операционной системой при выполнении файловых операций. Например, значение атрибута **Dir** позволяет отличить файл от подчиненного каталога, а по значению атрибута **Arc** отбираются файлы для резервного копирования, атрибут **R/O** блокирует возможность изменения и удаления файла, а атрибут **Hid** делает файл «невидимым» (команда DIR не выводит имя этого файла в списке). Если в записи корневого каталога установлено единичное значение атрибута **Vol**, то поля «Имя» и «Расширение» этой записи (максимум 11 байтов) будут трактоваться как метка тома, а все остальные данные этой записи будут игнорироваться.

Время и *дата* создания файла описываются двухбайтовыми блоками данных в специальных форматах.

Номер начального кластера – это та самая ссылка, с которой начинается сканирование FAT в процедуре поиска номеров кластеров, выделенных файлу при его записи на диск.

Размер файла – числовая характеристика файла, определяемая объемом хранимых в файле данных. Учитывая тот факт, что файлу при его записи на диск выделяется целое число кластеров, его размер практически всегда будет меньше, чем размер занятого им дискового пространства.

6.2.2 Алгоритмы выполнения типовых файловых операций

Стандартный набор операций над объектами файловой системы (дисками, каталогами и файлами) уже обсуждался выше в п. 6.1, все эти операции доступны пользователям Windows-систем через множество «контекстных меню», а пользователям MS DOS – через интерфейс «командной строки». Независимо от реализации пользовательского интерфейса, файловые операции выполняются соответствующими программными компонентами ОС, доступными также и прикладным программам через вызовы соответствующих системных функций.

Программная реализация любого алгоритма требует определенной организации обрабатываемых данных. Системные функции, реализующие файловые операции, используют структуры данных, сформированные в служебных областях дисковых устройств (*MBR, Boot Record, FAT* и *Root Directory*), подчиненные каталоги, хранимые в рабочей области в файлах специального формата, а также служебные структуры данных, организованные в CMOS-памяти. Для ускорения работы файловых функций дисковые структуры данных копируются (и регулярно обновляются) в специальные буферы оперативной памяти, количество которых определяется соответствующей командой системного конфигурационного файла CONFIG.sys (п. 5.2.1).

Обсуждая алгоритмы работы файловых функций, будем предполагать, что этим функциям доступны все структуры данных файловой системы, и при вызове функции ей передается текстовая строка, содержащая соответствующую исходную информацию – как правило, это имена объектов файловой системы и различные модификаторы. При этом неважно, введена ли эта информация пользователем в «командной строке» для ее последующей обработки интерпретатором COMMAND.com, программно сформирована приложением «контекстное меню» графического пользовательского интерфейса или сгенерирована прикладной программой – в любом случае она будет передана файловой системе через программный интерфейс функций DOS.

6.2.2.1 Алгоритм активизации тома

Для активизации тома требуется загрузить в ОЗУ структуры данных из его системной области, что сделает этот диск оперативно доступным.

Очевидно, это самая алгоритмически простая из файловых операций, для выполнения которой пользователю достаточно ввести в командной строке имя тома с двоеточием в конце, например, D: или F:, или выбрать диск из списка.

Соответствующая DOS-функция, получив имя тома, который следует активизировать, определяет тип устройства (сменный или фиксированный) и адрес порта ввода-вывода, связанного с этим устройством.

Если получено имя сменного устройства, на котором при форматировании создается единственный том, достаточно загрузить в буфер ОЗУ его нулевой *boot*-сектор, определить по таблице параметров форматирования номера секторов системной области, занятых таблицей FAT и корневым каталогом, и загрузить в буфер ОЗУ эти системные структуры данных.

Если полученное имя идентифицирует один из разделов фиксированного диска, алгоритм немного усложняется: обращением к *таблице разделов* диска (которая уже скопирована в буфер ОЗУ в процессе начальной загрузки ОС) определяется адрес *boot*-сектора этого раздела и далее – аналогично описанному выше алгоритму для сменного запоминающего устройства.

6.2.2.2 Алгоритм поиска файла

Алгоритм поиска файла «вложен» в алгоритмы реализации многих файловых функций – прежде, чем файл прочитать, скопировать, удалить или выполнить, надо, как минимум, его найти в файловой системе, то есть определить последовательность номеров кластеров, занятых этим файлом на томе (или установить факт отсутствия искомого файла).

Алгоритм поиска файла проиллюстрируем на примере команды TYPE, используемой для вывода на экран монитора содержимого текстовых файлов ASCII-формата, при этом процесс визуализации текста в уже «найденном» файле рассматривать не будем.

Задача заключается в том, чтобы найти файл, «путь» к которому задан соответствующим значением параметра команды, например, такой:

```
TYPE \my_docs\my_file.txt.
```

Единственный параметр команды TYPE (часть текста командной строки, расположенная правее первого пробела) – это *путь к файлу*, содержащий список имен подкаталогов с именем искомого файла в конце списка.

Строка «путь к файлу» формируется по следующим правилам:

- имена каталогов располагаются в списке слева направо в порядке их подчиненности: от родительских каталогов к дочерним;
- в качестве разделителя имен в списке используется символ «\»;
- если строка начинается с символа «\» (как в нашем примере), то путь считается заданным от корневого каталога текущего тома;
- если строка начинается с двух символов «точка» (.), то путь считается заданным от каталога, являющегося родительским для каталога, имя которого указано первым;
- в остальных случаях путь считается заданным от текущего каталога, «открытого» предшествующими командами.

0-й шаг алгоритма (для любой файловой операции) – это интерпретация строки параметров команды. В случае с командой TYPE параметр всего один, для более сложных команд параметры в строке (если их несколько) разделены символом «пробел». В дальнейшем при описании алгоритмов реализации файловых операций этот «нулевой» шаг алгоритма будем опускать, предполагая известными все компоненты «пути к файлу» – имена подкаталогов в порядке их подчиненности.

В результате интерпретации строки с параметром команды получим:

- родительский подкаталог файла подчинен корневному каталогу тома (так как текущим каталогом по условию нашей задачи является корневой каталог);
- имя родительского подкаталога искомого файла – my_docs;
- имя искомого файла – my_file;
- расширение имени файла – txt.

1-й шаг алгоритма – поиск текущего каталога, то есть каталога, «открытого» предшествующими командами, от которого и начинается путь к файлу, заданный в команде. Как правило, открытый каталог (или только первый кластер, выделенный этому каталогу) уже загружен в соответствующий буфер ОЗУ, и «искать» его на диске не требуется, так как он ранее был найден процедурой, рекурсивно использовавшей описываемый здесь алгоритм, начиная от корневого каталога активного диска.

Адрес корневого каталога (номера его начального и конечного секторов системной области диска) достаточно просто определяются по данным таблицы параметров форматирования тома (таблица 6.2), гарантированно загруженной в буфер ОЗУ в составе нулевого *boot*-сектора активного тома.

2-й шаг алгоритма – сканирование текущего каталога (таблица 6.4) с целью поиска объекта, подчиненного текущему каталогу. В рассматриваемом примере объект поиска – файл с именем *my_docs* (без расширения), являющийся подчиненным каталогом (4-й бит байта атрибута равен единице). Если такого объекта не найдено, поиск будет прекращен с выводом диагностического сообщения, в противном случае – запоминается *номер начального кластера* из множества кластеров, занятых объектом.

3-й шаг алгоритма – формирование списка номеров кластеров, занятых подкаталогом *my_docs*, путем сканирования таблицы FAT (она тоже загружена в буфер ОЗУ), начиная с элемента, равного номеру начального кластера, и далее по ссылкам, пока не будет найден элемент с кодом *EOF* (таблица 6.3).

4-й шаг алгоритма – преобразование списка номеров кластеров в список номеров секторов и передача этого списка контроллеру диска для загрузки содержимого этих секторов в буфер ОЗУ.

Далее рекурсивно выполняются шаги со 2-го по 4-й до тех пор, пока не будет найден объект, имя которого указано последним в строке «путь к файлу».

6.2.2.3 Алгоритм переименования файла

В команде – два параметра (две разделенных пробелом текстовых подстроки): первый параметр – это путь к переименовываемому файлу, а второй – новое имя файла, например: *RENAME \my_docs\my_file.txt new_name.doc*.

Вначале запускается рассмотренный выше алгоритм поиска файла – с той разницей, что остановить поиск надо чуть раньше, так как для переименования файла его содержимое не требуется, достаточно «найти» только его имя.

Если при сканировании подкаталога *my_docs* будет найдена регистрационная запись объекта с именем *my_file* и с расширением *txt*, и при этом переименование объекта не запрещено (то есть 0-й бит байта атрибутов объекта будет иметь нулевое значение), то поля «имя» и «расширение» в регистрационной записи будут заменены на новые, в нашем примере – на *new_name* и *doc*.

6.2.2.4 Алгоритм поиска свободного пространства на томе

Рассмотрим два варианта использования этого алгоритма: в первом случае алгоритм используется для оценки общего объема свободного пространства тома, а во втором – для поиска свободных кластеров, необходимых для записи на том файла заданного размера.

В обоих случаях в основе алгоритма – процедура сканирования таблицы FAT, определяющая количество (и, соответственно, номера) элементов FAT, содержащих нулевые значения, так как каждый такой элемент определяет свободный кластер рабочей области тома. Зная размер кластера в байтах (таблица 6.2), легко определяется и размер свободного пространства.

Для оценки общего объема свободного пространства тома сканируется *вся таблица FAT* и определяется количество ее нулевых элементов, которое затем умножается на размер кластера в байтах.

Во втором случае известен размер копируемого файла (в байтах), и требуется определить список номеров кластеров, количество которых будет минимально достаточным для записи этого файла на том. Алгоритм может быть реализован следующей последовательностью шагов:

1-й шаг – вычисляется количество требуемых кластеров (делением размера файла на размер кластера с округлением до большего целого).

2-й шаг – последовательным просмотром таблицы FAT определяется необходимое количество нулевых элементов с сохранением списка номеров найденных элементов.

3-й шаг – в каждом (кроме последнего) элементе сформированного списка вместо нуля записывается номер следующего за ним элемента, а в последнем элементе записывается код окончания файла FFF_h (EOF).

4-й шаг – сохраняется номер первого элемента списка для его последующей вставки в новую регистрационную запись родительского каталога (поле «номер начального кластера», таблица 6.4).

Так как FAT обеспечивает возможность фрагментарного хранения файла, существует множество вариантов реализации 2-го шага рассмотренного выше алгоритма. В ранних версиях ОС, когда ПК оснащались дисковыми накопителями малой емкости, главным критерием качества алгоритма была эффективность использования дискового пространства. В этих условиях функции поиска свободных кластеров использовали «скупой»²² алгоритм, который, к тому же, самый экономичный в реализации: в процессе сканирования таблицы FAT, начиная с ее первого элемента, номер каждого нулевого элемента FAT сохраняется в списке номеров кластеров, пока длина этого списка не станет достаточной.

²² Намеренно не используем здесь слово «жадный», так как название «жадный алгоритм» (*greedy algorithm*) уже занято – этим термином принято обозначать определенную группу алгоритмов принятия оптимальных решений.

Такой алгоритм минимизирует количество мелких фрагментов свободных кластеров, но увеличивает количество фрагментарно хранимых файлов, что снижает производительность дисковых операций и вынуждает пользователя ПК чаще проводить операцию дефрагментации дискового пространства.

Другие, менее «скупые», алгоритмы поиска свободных кластеров, при сканировании таблицы FAT пренебрегают теми непрерывными фрагментами свободных кластеров, длины которых недостаточны для хранения всего файла, и формируют список номеров кластеров из состава кластеров, принадлежащих какому-то одному крупному фрагменту. При отсутствии крупных фрагментов выполняются различные модификации «скупого» алгоритма, минимизирующие количество непрерывных фрагментов кластеров, выделяемых файлу.

6.2.2.5 Алгоритм удаления файла

Какую цель преследует пользователь ПК, удаляя файл²³, и какими будут последствия успешного завершения этой процедуры? Очевидно, что главная цель связана с необходимостью освобождения места на томе²⁴ для записи на него других файлов, а видимых пользователю последствий удаления файла всего два: во-первых, стандартные утилиты просмотра каталогов (например, DOS-команда *DIR*) не показывают удаленные файлы в списках объектов их родительских каталогов, и, во-вторых, свободного места на томе стало больше, причем ровно на то количество кластеров, которое занимал удаленный файл (в чем легко убедиться, анализируя свойства тома до и после удаления файла).

При рассмотрении алгоритма удаления файла следует иметь в виду, что технологии записи двоично-кодированной информации на компьютерные носители не требуют физического удаления данных с носителя перед тем, как на их место будут записаны новые, также двоично-кодированные, данные.

И еще одна деталь, уже не технологического а, скорее, психологического характера: кто из нас не попадал в ситуацию, когда удаленный вчера файл, казавшийся нам совершенно ненужным, уже сегодня оказывается жизненно необходимым и требующим оперативного восстановления (и при этом, желательно – без потери полезной информации)?

²³ Здесь не идет речь о привычной современному пользователю «корзине» – специально организованной папке, предназначенной для временного хранения удаленных файлов.

²⁴ Разумеется, могут быть и другие цели, например, связанные с необходимостью удаления файлов с конфиденциальной информацией или с истечением установленных сроков хранения деловой документации.

Очевидно, примерно так была поставлена задача разработчикам алгоритма удаления файла, реализованного ими при разработке файловой функции, связанной с DOS-командой *DELeTe* «*путь-к-удаляемому-файлу*».

Алгоритм реализуется следующими последовательными шагами:

1-й шаг – Выполняется рассмотренный выше алгоритм поиска удаляемого файла, в результате которого отыскивается регистрационная запись этого файла в родительском каталоге и определяется список элементов таблицы FAT, представляющий последовательность кластеров, занятых файлом.

2-й шаг – Все элементы таблицы FAT, входящие в список номеров кластеров, занятых удаляемым файлом, заполняются нулевыми значениями – тем самым соответствующие кластеры рабочей области тома объявляются свободными для последующей записи в них новых файлов.

3-й шаг – Регистрационная запись удаленного файла в родительском каталоге специальным образом *маркируется* – первый из восьми символов поля «имя файла» заменяется специальным символом с ASCII-кодом E5_h²⁵.

ASCII-код E5_h определяет один из символов дополнительной кодовой таблицы, которые, согласно правилам MS DOS, нельзя было использовать для именования файлов. Наличие маркированных записей позволяет программам просмотра каталогов игнорировать все записи об удаленных файлах, а программам восстановления удаленных файлов – наоборот, выбирать только те записи, которые начинаются с такого «специального» символа.

Заметим, что *файл физически не удаляется* из занятых им кластеров, и запись об удаленном файле также *не удаляется* из родительского каталога, что делает доступной всю ее информацию, включая размер файла, время его создания, расширение и последние 7 символов имени файла, а также номер начального кластера. Все это сохраняет потенциальную возможность восстановления удаленного файла – по крайней мере, до тех пор, пока условно освобожденные этим файлом кластеры не будут заняты каким-либо другим файлом.

Выбор способа маркировки регистрационных записей удаленных файлов путем модификации имен файлов кажется труднообъяснимым – почему бы не использовать для этой цели один из двух свободных битов байта атрибутов?

²⁵ Причину, по которой для маркировки удаленных файлов был выбран символ с ASCII-кодом E5_h, а не какой-то другой из 128 символов дополнительной кодовой таблицы, теперь уже вряд ли удастся определить достоверно, остается только предполагать, что в команде разработчиков MS DOS были русскоязычные программисты, не лишенные чувства юмора.

К тому же, такая маркировка создала проблему при переходе на новое соглашение об именовании файлов, разрешившее использовать в именах символы национальных алфавитов. Так как ASCII-код E5_h определяет строчную букву алфавита, пришлось все строчные символы имени файла преобразовывать в соответствующие прописные²⁶, чтобы файлы с именами, начинающимися со ставшего разрешенным символа с кодом E5_h, не получали статус удаленных файлов в момент их создания.

6.2.2.6 Алгоритм создания подчиненного каталога

Подчиненный каталог – это промежуточный узел иерархической (древовидной) структуры данных, непосредственно связанный с единственным *родительским узлом* (корневым или другим подчиненным каталогом) и множеством *дочерних узлов* (подчиненных каталогов или файлов).

По своей структуре подчиненный каталог подобен корневому – он содержит множество 32-байтовых записей, используемых для регистрации дочерних объектов, но, в отличие от корневого, подчиненный каталог является файлом специального формата и хранится, как и все другие файлы, в рабочей области тома, занимая в ней определенную последовательность кластеров. Количество кластеров, занятых подчиненным каталогом, динамически изменяется и в любой момент времени должно быть минимально достаточным для регистрации всех его дочерних объектов.

Для создания подчиненного каталога используется DOS-команда MD (*Make Directory*), единственным параметром которой является «путь», заканчивающийся именем создаваемого каталога: например, в результате выполнения команды **MD docs\my_docs** будет создан каталог **my_docs**, подчиненный своему родительскому каталогу **docs** (который, в свою очередь, подчинен каталогу, установленному текущим к моменту выполнения команды).

На первом шаге реализуется *алгоритм поиска объекта* – родительского каталога **docs**, и в нем создается новая регистрационная запись для дочернего объекта **my_docs** (если, разумеется, в этом каталоге еще нет дочернего объекта с таким же именем). В этой записи заполняются все поля (в том числе – устанавливается атрибут *dir*, позволяющий отличить файлы типа «каталог» от файлов других типов), кроме адресного поля – пока еще неизвестного *номера начального кластера*.

²⁶ Алгоритм преобразования «длинных» Windows-имен файлов в «короткие» DOS-имена обсуждался выше в п.6.2.1 при рассмотрении структуры записей корневого каталога.

Следующий шаг алгоритма – *поиск свободных кластеров*, необходимых для хранения создаваемого подкаталога *my_docs*. В момент создания каталога ему выделяется единственный кластер, номер этого кластера записывается в адресном поле регистрационной записи родительского каталога, а в соответствующем элементе таблицы FAT записывается код FFF_h (EOF).

На завершающем этапе алгоритма оформляется содержимое кластера, выделенного каталогу.

Во-первых, кластер «обнуляется» – в каждый из его байтов записывается ноль, в результате чего физически удаляются все данные, которые ранее были записаны в этом кластере (это могли быть данные удаленных файлов, ранее занимавших этот кластер, или данные программы форматирования тома, тестирующей каждый сектор путем записи/чтения специальных кодовых последовательностей).

Во-вторых, в кластере создаются две служебные записи для фиктивных объектов типа «каталог» (их атрибут *dir* установлен в единицу). Эти записи подобны другим записям каталогов, за исключением имен объектов:

- первый фиктивный объект получает имя «.» (точка), а в качестве адресной ссылки этого объекта указывается номер кластера, выделенного создаваемому каталогу («ссылка на себя»);

- имя второго фиктивного объекта – «..» (две точки, не путать с двоеточием), а в поле адресной ссылки этого объекта указывается номер начального кластера родительского каталога («ссылка на родителя»); если родительским является корневой каталог, то в поле ссылки указывается ноль (что допустимо, так как число «ноль» не используется для нумерации кластеров).

Такие «точечные» имена не могут быть именами реальных объектов файловой системы, они выполняют роль маркеров служебных записей, используемых для оптимизации алгоритмов поиска и просмотра подчиненных каталогов.

Казалось бы, зачем хранить ссылку на начальный кластер каталога в самом этом начальном кластере? Причина – во фрагментарности расположения файлов, причем вероятность фрагментарного расположения подчиненных каталогов существенно выше, чем обычных файлов.

Покажем это на простом примере: пусть на томе достаточно много непрерывных фрагментов свободных кластеров, тогда при создании каталога ему будет выделен один кластер в начале первого из таких фрагментов. Далее в этот каталог массово копируются файлы.

Если кластер имеет размер 1024 байта (в нашем примере), а каждая регистрационная запись занимает 32 байта, одного кластера каталога хватит для регистрации только первых 32-х дочерних объектов (точнее, только первых 30, так как в каталоге уже созданы две служебные записи, а фактически – еще меньше, если в каталог копируются объекты с «длинными» Windows-именами, для хранения каждого из которых может потребоваться от одной до восьми дополнительных 32-байтовых записей).

Для продолжения копирования файлов потребуется расширение каталога, и будет запущен алгоритм поиска еще одного свободного кластера, в котором можно будет зарегистрировать еще 32 дочерних объекта, и так далее до тех пор, пока не будет исчерпан список копируемых файлов. Какова вероятность того, что при очередном расширении каталога ему будет выделен кластер с номером, на единицу большим номера предыдущего кластера? Очевидно, что близкая к нулю, так как следующие (после начального) кластеры, бывшие свободными в момент создания каталога, к моменту его расширения с высокой вероятностью уже будут заняты дочерними объектами этого каталога.

А теперь представим себе реализацию алгоритма поиска дочернего объекта (п. 6.2.2.2) в таком сильно фрагментированном каталоге. На предыдущем шаге алгоритма по ссылке из родительского каталога был найден и загружен в буфер ОЗУ начальный кластер этого каталога, после чего запускается процедура его сканирования с целью поиска дочернего объекта. Если в начальном кластере каталога искомого объекта нет, потребуется найти следующий кластер каталога, затем, возможно, еще один и так далее по всей последовательности номеров кластеров, заданной в таблице FAT для этого каталога.

Проблема в том, что ссылка на начало этой последовательности хранится в родительском каталоге, и сохранение ее дубликата в виде «ссылки на себя» в начальном кластере каталога (который гарантированно загружен в буфер ОЗУ) позволяет исключить операцию чтения родительского каталога при поиске дочерних объектов.

Если «ссылка на себя» позволяет оптимизировать алгоритм поиска дочерних объектов (спуск по дереву каталогов), то «ссылка на родителя» во второй служебной записи подчиненного каталога позволяет оптимизировать алгоритм поиска родительского каталога (подъем на один уровень по дереву каталогов), исключая необходимость доступа к родительскому каталогу своего родительского каталога для поиска соответствующей адресной ссылки.

Например, DOS-команда *CD* (*Change Directory*) применяется для открытия каталога, путь к которому задается единственным параметром этой команды. Алгоритм поиска элементов заданного «пути» (п. 6.2.2.2) будет последовательно отыскивать в каталогах имена дочерних объектов и соответствующие адресные ссылки. В команде вида *CD ..* путь включает имя фиктивного объекта «*..*», регистрационная запись которого в текущем каталоге содержит ссылку на его родительский каталог.

Следующие три примера также иллюстрируют полезность хранения ссылки на родителя непосредственно в текущем каталоге.

В результате выполнения DOS-команды *MD ..\my-brother-dir* будет создан каталог, подчиненный родителю текущего каталога (то есть каталог одного уровня с текущим), команда *COPY ..\old_file new_file* создаст в текущем каталоге копию файла родительского каталога, а команда *COPY old_file ..\new_file* – наоборот, скопирует файл из текущего каталога в родительский.

6.2.2.7 Алгоритмы копирования и перемещения файлов

Операция копирования файла реализуется последовательным выполнением рассмотренных выше алгоритмов поиска файла (п. 6.2.2.2) и поиска свободного пространства для записи файла (п. 6.2.2.4). В результате создается копия исходного файла в том же или другом каталоге.

Операция перемещения файла реализуется в двух вариантах, в зависимости от того, куда перемещается файл: на другой том, или в другой каталог того же тома. В первом случае файл сначала копируется, как это описано выше, а затем удаляется по алгоритму, описанному в п.6.2.2.5. Второй вариант не предусматривает создания копии исходного файла – файл остается в тех же кластерах, но его регистрационная запись перемещается в другой каталог.

6.2.2.8 Алгоритмы восстановления удаленных файлов

Анализ рассмотренных выше алгоритмов записи и удаления файлов показывает, что в ряде случаев сохраняется возможность восстановления удаленного файла (до тех пор, пока на его место физически не записан другой файл).

Для восстановления логически удаленного файла достаточно *изменить в регистрационной записи каталога первый символ его имени на любой из допустимых символов*, что сделает «видимой» запись об удаленном файле, и *восстановить в таблице FAT список номеров кластеров, занятых этим файлом*, что даст возможность доступа к его содержимому.

Проблема в том, что все элементы этого списка таблицы FAT при удалении файла получили нулевые значения и стали формально неотличимыми от множества других элементов FAT, представляющих свободные кластеры (в том числе и те, которые ранее были заняты впоследствии удаленными файлами).

В этих тяжелых для программиста условиях определенный оптимизм вселяет тот факт, что процедура удаления файла только маркирует его регистрационную запись, не удаляя ее из соответствующего каталога, что делает доступной информацию об удаленном файле:

- если регистрационная запись об удаленном файле присутствует в каталоге (в чем можно убедиться по последним семи символам его имени, так как имена всех удаленных файлов начинаются одинаково – с символа с ASCII-кодом E5_h), значит можно продолжить попытку его восстановления, так как в этом случае начальный кластер удаленного файла еще не занят другими файлами, записанными на диск после того, как этот файл был удален;
- сохранена адресная ссылка (номер начального кластера), то есть известно, откуда следует начинать поиск номеров кластеров, занятых файлом;
- известен размер удаленного файла, что позволяет вычислить количество занятых им кластеров;
- известны дата и время создания удаленного файла, что может быть полезным в случае наличия на томе множества других удаленных файлов (время создания которых также известны), в том числе и не требующих восстановления;
- известен алгоритм поиска свободного дискового пространства (п.6.2.2.4) для записи файла, реализованный соответствующей системной функцией файловой системы.

Очевидно, что главная проблема восстановления удаленных файлов связана с фрагментарностью их расположения, поэтому для повышения надежности работы процедур восстановления рекомендуется периодически дефрагментировать диск с помощью специальных программных утилит, которые перераспределяют дисковое пространство, закрепляя за файлами последовательности из непрерывно расположенных кластеров. Процедура дефрагментации диска предполагает создание временных копий файлов, требует наличия на диске свободного пространства и является весьма затратной по времени.

6.2.3 Недостатки FAT-систем

FAT-системы успешно применялись многие годы в составе различных ОС, прежде всего – в DOS и Windows. Первоначально, они создавались для НГМД – накопителей на гибких магнитных дисках (первыми такими устройствами были 8-дюймовые односторонние гибкие диски информационной емкостью 160 Кб), и одним из основных требований, предъявляемых к файловой системе в те годы, была экономичность хранения служебной информации.

Следует отметить, что разработчики системы FAT-12 успешно справились с реализацией этого требования, и основные концепции, заложенные при ее создании, оказались работоспособными и в применении к жестким дискам существенно большей (по сравнению с НГМД) емкости. Правда, это потребовало увеличения разрядности таблицы FAT вначале до 16, а затем и до 32-х битов, что неизбежно привело к увеличению объема хранимых на диске системных данных.

Со временем состав требований, предъявляемых к файловым системам, существенно расширился, в новых условиях FAT-системы перестали справляться с задачами управления файлами, а отдельные технологические и алгоритмические решения этих систем потеряли свою актуальность. Тем не менее, и сегодня FAT-системы вполне успешно используются на персональных компьютерах – в основном, для временного хранения информации на сменных твердотельных накопителях небольшой емкости.

Основная критика в адрес FAT-систем нацелена на те их недостатки, которые связаны с организацией хранения и доступа к файлам.

1) *Неэффективное использование дисковой памяти.*

При форматировании дисков большой емкости приходится решать нетривиальную задачу выбора размера кластера: большие размеры кластеров уменьшают их количество, соответственно уменьшается размер таблицы FAT и увеличивается рабочая область тома, но при этом снижается эффективность хранения небольших файлов; уменьшение размеров кластеров позитивно сказывается на эффективности использования рабочей области тома, но приводит к уменьшению ее общего размера.

2) *Низкая эффективность таблицы FAT.*

Таблица FAT – это основная структура данных, используемая алгоритмами поиска файлов и алгоритмами поиска свободного пространства. Казавшийся когда-то эффективным способ хранения такой разнородной информации в единой структуре данных фиксированного размера, с увеличением разрядности FAT

стал проявлять свои негативные стороны. Хранение в элементах FAT и адресной информации (номера «следующих» кластеров файлов), и информации о свободных кластерах (число «0») привело к тому, что, например, в FAT-32 для хранения числа «0» расходуется *8 байтов* вместо *одного бита*, которого вполне хватило бы для того, чтобы отличить занятый кластер от свободного. В результате, размер таблицы FAT возрастает и остается максимальным независимо от количества занятых кластеров рабочей области.

3) *Существенная фрагментация файлов.*

При длительной эксплуатации дисковых накопителей операции удаления файлов (так же, как и операции редактирования, уменьшающие их размеры) объективно приводят к фрагментации свободного пространства, а используемые алгоритмы поиска свободного пространства при копировании (или увеличении размеров) файлов способствуют их фрагментации. В результате снижается быстродействие системы и увеличивается износ оборудования.

4) *Ограничение длины имени файла.*

Ограничение длины имени файла восемью байтами при его хранении в соответствующем поле записи каталога, приводит к необходимости дублирования записей и снижает эффективность хранения подчиненных каталогов, способствуя увеличению их фрагментарности.

5) *Низкая надежность хранения данных.*

Надежность работы любой технической системы оценивается по двум основным параметрам: по длительности работы без сбоев (так называемая «наработка на отказ») и по времени восстановления системы после сбоя.

Разработчики FAT-систем предусмотрели различные средства повышения надежности работы файловой системы, однако эффективность этих средств явно не соответствует современным требованиям.

В ситуациях с «жестким сбоем», связанных с разрушением носителя информации (например, магнитного слоя рабочей поверхности диска):

- фрагментарность файлов, приводящая к преждевременному износу оборудования, косвенно уменьшает время наработки на отказ;
- дублирование таблицы FAT, основного «адресного справочника» файловой системы, снижает время восстановления системы, так как при утере основной FAT вся адресная информация может быть восстановлена по ее зеркальной копии; однако следует учесть, что обе FAT размещены в си-

стемной области тома, занимая в ней соседние секторы, поэтому вероятность того, что при разрушении основной FAT ее зеркальная копия останется доступной, крайне низка.

В ситуациях с «мягким сбоем», не связанных с разрушением носителя информации, например, при ошибочном удалении файла:

- FAT-системы позволяют блокировать попытки ошибочного удаления особо ценных файлов путем установки в «1» атрибута *Read Only* в регистрационных записях этих файлов в соответствующих каталогах, однако такой прием одновременно заблокирует возможность выполнения ряда других файловых операций;
- FAT-системы сохраняют возможность восстановления удаленных файлов, но они ограничиваются фрагментарностью расположения файлов и по времени восстановления не конкурируют с «корзиной» Windows.

б) *Отсутствие средств разграничения доступа к файлам.*

В то время, когда разрабатывались FAT-системы, проблема информационной безопасности не стояла так остро, как сегодня, и поэтому средства защиты от несанкционированного доступа к файлам проработаны в этих системах весьма слабо. Защита от несанкционированного копирования или модификации файла обеспечивается установкой в «1» его атрибута *Read Only*. Для защиты файла с конфиденциальной информацией от несанкционированного просмотра можно установить в «1» его атрибут *Hidden*, что придаст ему статус «скрытого файла», однако, во-первых, это не *разграничение*, а *ограничение доступа*, и, во-вторых, такие ограничения легко обходятся каждым, кто знаком с DOS-командой *attrib*.

6.3 Файловые NTFS-системы

По сравнению с FAT-системами файловая система NTFS (*New Technology File System*) обладает лучшими показателями по производительности и надежности, а также обеспечивает полнофункциональную защиту данных, позволяя гибко управлять доступом к данным и разграничивать доступ пользователей как к отдельным файлам, так и к каталогам.

NTFS поддерживает объектно-ориентированную файловую модель, рассматривая все файлы как объекты, которые имеют определяемые пользователем и системой атрибуты-свойства. Минимальный набор стандартных атрибутов файлов приведен в таблице 6.5. Форматы атрибутов и способы их хранения рассматриваются в п. 6.3.5.

Таблица 6.5 – Минимальный набор стандартных атрибутов файлов

Имя атрибута	Назначение атрибута	
<i>H</i>	Header	Заголовок MFT-записи (содержит, в частности, ее номер.
<i>SI</i>	Standard Information	Стандартная информация.
<i>FN</i>	File Name	Имя файла.
<i>Data</i>		Данные файла.
<i>SD</i>	Security Descriptor	Дескриптор безопасности.

Понятие *атрибута файла*, которое в FAT-системах ограничивалось фиксированным набором из шести бинарно-кодируемых свойства, в NTFS существенно расширено: в частности, наряду со стандартными атрибутами появилась возможность создания пользовательских атрибутов произвольного назначения.

6.3.1 Структура тома NTFS

NTFS, так же, как и FAT, использует понятие кластера – минимального блока, выделяемые файлам при их записи на том. В отличие от FAT, NTFS делит на кластеры *все пространство тома*, в том числе и его системную область, называемую в NTFS *MFT-зоной*. При этом поддерживаются кластеры практически любых размеров – от 512 байт до 64 Кб, стандартом считается размер 4 Кб.

Том NTFS условно делится на две зоны: примерно 12% кластеров, расположенных в начале тома, отводятся под так называемую *MFT-зону*, в которой располагается самый главный служебный файл MFT (*Master File Table – главная таблица файлов*), а остальные примерно 88% кластеров образуют рабочую зону, используемую для хранения как служебных, так и пользовательских файлов. Запись в MFT-зону других файлов, кроме самого MFT, запрещена – это сделано для того, чтобы исключить фрагментацию MFT при его расширении.

При этом не все так просто с MFT-зоной: свободными для записи считаются все свободные кластеры – в том числе и кластеры MFT-зоны, не занятые MFT-файлом. Если объем рабочей зоны становится недостаточным для записи файлов, MFT-зона сокращается (в два раза), предоставляя рабочей зоне дополнительные кластеры, а при освобождении рабочей зоны MFT-зона может снова расширяться. При этом не исключена ситуация, когда в этой зоне остались и обычные файлы – файл MFT в этом случае может фрагментироваться.

6.3.2 Мета-файлы NTFS

Главный «лозунг» файловой модели NTFS: «*Файлы, и ничего, кроме Файлов*». **Любой** элемент файловой системы представляет собой файл, в том числе и элементы, содержащие служебную информацию: корневой каталог, загрузочная

запись, журнал транзакций, список прав пользователей и другие, так называемые *мета-файлы*.

Состав метафайлов (таблица 6.6), их структура и назначение фиксированы, их имена начинаются с символа \$, и все они в обязательном порядке регистрируются в корневом каталоге, который сам также является метафайлом.

Таблица 6.6 – Метафайлы NTFS

Идентификатор файла	Имя мета-файла	Назначение файла
0	\$MFT	Сам MFT
1	\$MFTmirr	Зеркальная копия первых 16 записей MFT
2	\$LogFile	Журнал транзакций, который используется для восстановления файловой системы после сбоев
3	\$Volume	Метка тома и прочая служебная информация
4	\$AttrDef	Список стандартных атрибутов файлов
5	\$	Корневой каталог
6	\$Bitmap	Битовая карта занятости кластеров: 0 – кластер свободен, 1 – кластер занят.
7	\$Boot	Загрузочная запись. Сдержит стандартный набор параметров BIOS, а также начальные номера кластеров основного MFT и его зеркальной копии.
8	\$BadClus	Список сбойных кластеров
9	\$Quota	Описание прав доступа пользователей
10	\$Upcase	Таблица соответствия Unicode-кодов заглавных и строчных букв в именах файлов.
11-15		Резерв

Основной мета-файл в системе NTFS – это MFT, централизованный каталог всех файлов тома (в некотором смысле MFT выполняет ту же роль, что и корневой каталог вместе с таблицей расположения файлов в FAT-системах).

MFT состоит из записей фиксированного размера (обычно 2 Кб), каждая из которых содержит информацию о каком-либо одном файле. Каждый файл (в том числе и сам MFT) представлен в MFT-файле, по крайней мере, одной записью, позиционный номер которой является уникальным *идентификатором файла*. Зная этот идентификатор, можно восстановить по ссылкам список номеров всех остальных MFT-записей этого файла и далее, также по ссылкам – список номеров всех кластеров, занятых этим файлом.

Первые 16 записей MFT содержат данные *метафайлов*, причем самая первая (нулевая) запись представляет сам MFT. Учитывая ценность информации, хранимой в метафайлах, зеркальная копия первых 16 записей MFT хранится в

отдельном метафайле \$MFTmirr, расположенном в рабочей зоне – примерно по середине дискового пространства тома (как видим, разработчиками NTFS учтен недостаток FAT-систем, в которых зеркальная копия таблицы FAT хранится в системной области, практически рядом с основной таблицей, что снижает надежность дисковой системы компьютера при жестких сбоях).

6.3.3 Схемы хранения файлов

Основные претензии к FAT-системам высказывались в связи с низкой эффективностью использования дискового пространства этими системами при хранении файлов радикально различных размеров, а также в связи с потерями производительности операций доступа к файлам по причине их фрагментации. В NTFS система хранения файлов оптимизирована по этим критериям, что позволило эффективно использовать кластеры небольшого размера без увеличения затрат на хранение адресной информации.

Во-первых, все файлы в NTFS классифицируются по размерным категориям, и для каждой категории используется своя система хранения, оптимальная как по критерию фрагментарности файлов, так и по критерию производительности выполнения операций их поиска.

Во-вторых, система хранения адресной информации о фрагментированных файлах (а фрагментация файлов больших размеров объективно неизбежна) сделана существенно более компактной по сравнению с таблицей FAT, в которой приходилось хранить упорядоченные списки номеров всех (кроме первого) кластеров, занятых файлом.

В-третьих, для поиска файлов в NTFS применена технология индексации, широко используемая в системах управления базами данных. Каталоги, которые в FAT-системах были представлены линейными списками имен файлов с адресными ссылками на начальные кластеры этих файлов, получили в NTFS статус индексов – иерархических структур данных, позволивших отказаться при поиске от операций полного перебора элементов списков в пользу операций «спуска по дереву», существенно ускоряющих поиск файлов по значениям их атрибутов.

Схема хранения *небольших (small)* файлов.

Если файл имеет небольшой размер (как правило, не более 1,5 Кб), он может получить статус *резидентного файла* и будет целиком располагаться «внутри» атрибута *Data* (рисунок 6.2) своей единственной MFT-записи. При этом возможны ситуации, когда файл малого размера не получает статус резидентного из-за того, что суммарный размер всех других его атрибутов (состав и

размеры которых могут быть переменными), не позволяет ему разместиться внутри одной MFT-записи длиной 2Кб.

Резидентные файлы не занимают кластеров в рабочей зоне тома, и доступ к ним осуществляется наиболее эффективно.

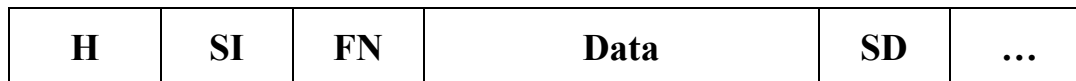


Рисунок 6.2 – Схема хранения небольших (Small) файлов

Схема хранения *больших (Large) файлов*.

Если файл не помещается внутри своего атрибута *Data*, он становится *нерезидентным* и располагается вне MFT-зоны, занимая в рабочей зоне один или более кластеров. При этом статус *резидентного* получает атрибут *Data* этого файла, который будет содержать уже не данные файла, а адресные ссылки на фрагменты кластеров рабочей зоны, занятых файлом (рисунок 6.3).

Каждая такая ссылка определяет один фрагмент непрерывно расположенных кластеров и содержит три параметра: *LN* – порядковый номер начального кластера фрагмента, *VN* – виртуальный номер начального кластера фрагмента (смещение начального кластера фрагмента относительно начала файла) и *CL* - количество кластеров во фрагменте.

Таким образом, содержимое атрибута *Data* начальной (и единственной) MFT-записи нерезидентного файла позволяет определить упорядоченный список номеров кластеров, занятых этим файлом.

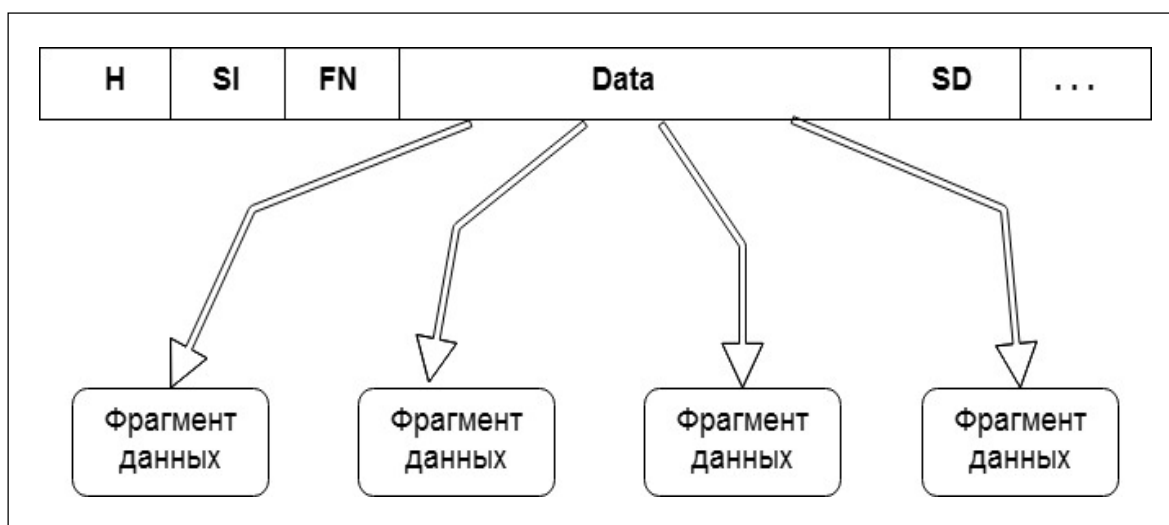


Рисунок 6.3 – Схема хранения больших (Large) файлов

Схема хранения *очень больших (Huge)* файлов.

Если файл настолько сильно фрагментирован, что его атрибут *Data* (со ссылками на фрагменты кластеров, занятых этим файлом) не помещается в начальной MFT-записи файла, то *нерезидентным* становится сам этот атрибут, и для его хранения резервируется вторая MFT-запись (рисунок 6.4), ссылка на которую помещается в начальную MFT-запись файла в качестве *внешнего атрибута EA (External Attribute)*. При этом нерезидентный атрибут *Data*, помещенный во вторую MFT-запись файла, содержит ссылки на фрагменты кластеров, занятых файлом, точно так же, как и в случае с *Large*-файлами.

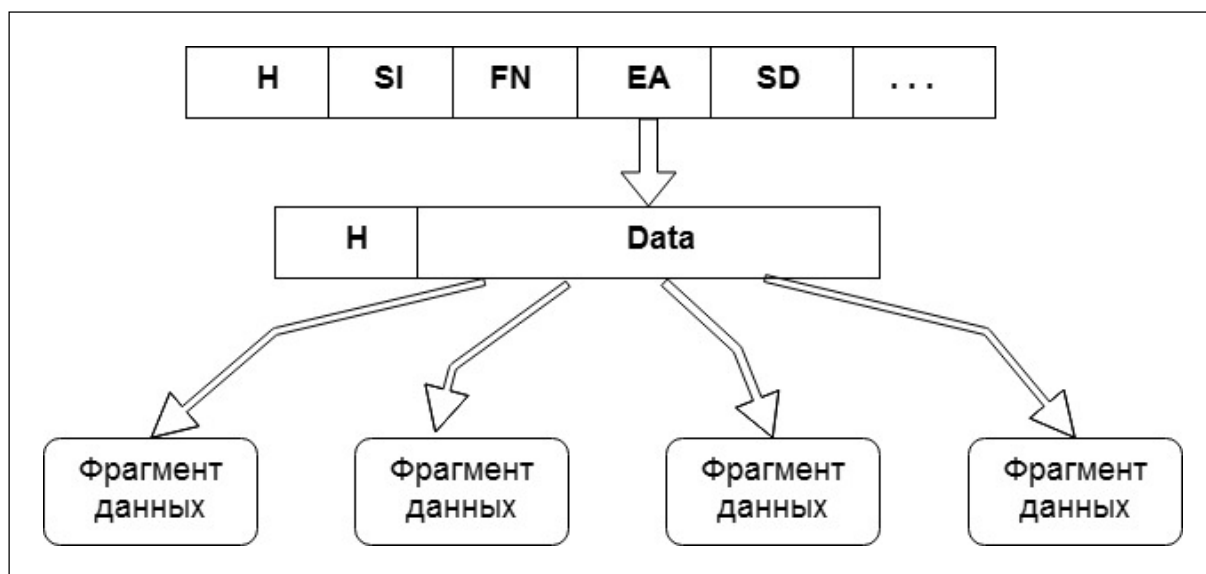


Рисунок 6.4 – Схема хранения очень больших (*Huge*) файлов

Схема хранения *сверхбольших (Extremely Huge)* файлов.

Если нерезидентный атрибут *Data* не помещается и во вторую MFT-запись файла, то внешний атрибут *EA*, хранимый в начальной MFT-записи файла, может содержать несколько упорядоченных ссылок на дополнительные MFT-записи этого файла, в каждой из которых хранится свой нерезидентный атрибут *Data* (рисунок 6.5) с множеством ссылок на фрагменты кластеров, занятых файлом.

Разумеется, в такой ситуации увеличивается размер внешнего атрибута *EA*, и когда ему перестанет хватать места в начальной MFT-записи файла, он также станет нерезидентным.

Заметим, что и многие другие атрибуты файла могут храниться в нерезидентной форме, что практически снимает ограничения на их размеры.

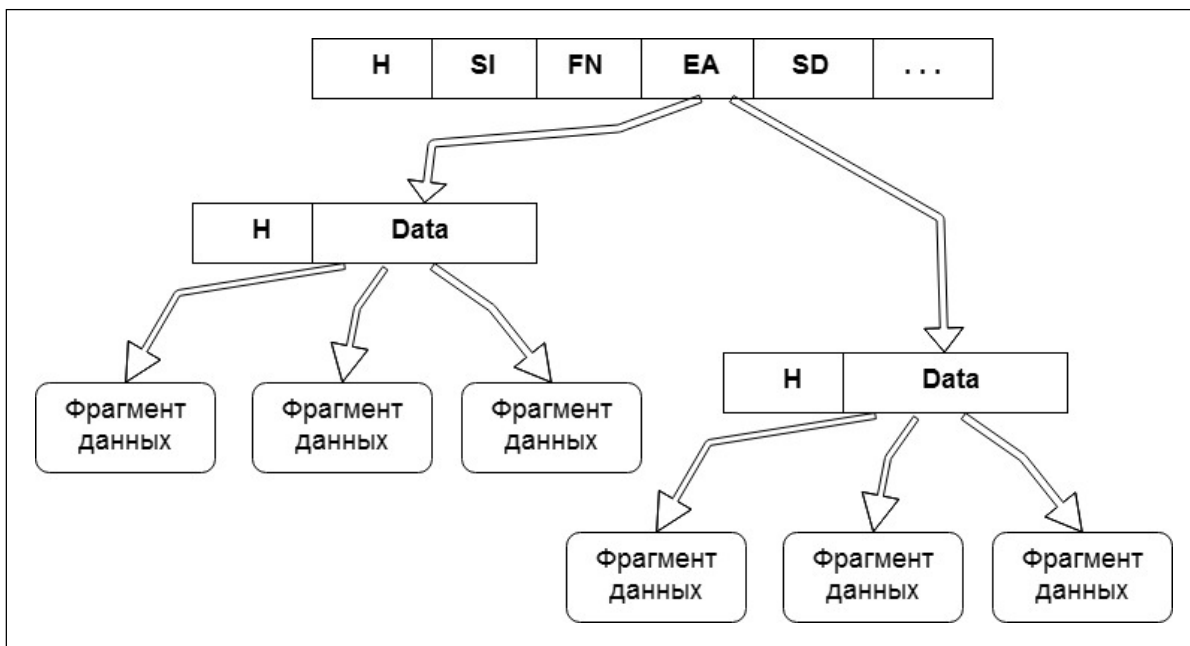


Рисунок 6.5 – Схема хранения сверхбольших (*Extremely Huge*) файлов

6.3.4 Схемы хранения каталогов

Как уже отмечалось, FAT-системы и NTFS по-разному трактуют понятие *каталога*. С позиций FAT-систем MFT-файл выполняет роль *дерева каталогов*, так как содержит регистрационные записи обо всех файлах со ссылками на номера занятых ими кластеров, или на другие MFT-записи (выполняющие роль подчиненных каталогов), в которых размещены такие ссылки. Процедура последовательного сканирования MFT-файла позволит найти любой файл по значению его атрибута *FN (File Name)* и по ссылкам определить расположение этого файла в рабочей зоне тома.

Однако, есть проблема, практически исключающая возможность прямого использования MFT-файла в качестве такого «суперкаталога», и эта проблема – в производительности процедуры полного сканирования. Как известно, файловая система NTFS создавалась для обслуживания накопителей очень большой емкости, и MFT-файл может достигать гигантских размеров (порядка 8 Тб при длине MFT-записи в 2 Кб и 4-байтовой длине поля номера такой записи), что потребует множества обращений к диску для его полного сканирования.

Для ускорения поиска файлов NTFS использует файлы специальной структуры, называемые *каталогами* или *индексными списками (Index List)*. В отличие от каталогов FAT-систем, NTFS-каталоги содержат только адресную информацию в форме ссылок на дочерние объекты – подчиненные им каталоги, файлы, или фрагменты кластеров тома.

Один такой индексный список – корневой каталог – является метафайлом, имеет фиксированное имя \$ (очевидно, разработчики NTFS решили подчеркнуть таким именем большую ценность информации, содержащейся в корневом каталоге) и хранится в одной из начальных записей MFT-файла. Остальные (подчиненные) каталоги хранятся, как и обычные файлы, и могут создаваться и именоваться пользователями стандартными средствами, например, командой MD в случае использования интерфейса командной строки.

Каждый каталог имеет одну запись в MFT-файле, которая содержит стандартный набор атрибутов (*H*, *FN*, *SI*, *SD*) и дополнительно – специальный атрибут *RI* (*Root Index – корневой индекс*), который, собственно, и является *индексным списком*, каждый элемент которого содержит пару: имя дочернего объекта (файла или подчиненного каталога) и адресную ссылку, в качестве которой используется номер начальной MFT-записи этого объекта. Завершается индексный список строкой «#####», используемой в качестве кода конца списка.

Длина списка *RI* каталога зависит от количества зарегистрированных в этом каталоге дочерних объектов и фактически определяет и размер всего каталога. Каталоги, как и файлы NTFS классифицируются по их размерам, и для каждой размерной категории каталогов применяется соответствующая схема их хранения, подобно тому, как это сделано для файлов.

Схема хранения *небольших (Small)* каталогов

Если количество дочерних объектов каталога невелико, то его атрибут *RI* получает статус *резидентного* и полностью размещается в начальной MFT-записи каталога (рисунок 6.6).

<i>H</i>	<i>SI</i>	<i>FN</i>	<i>RI</i> : <aaa; 232> <abcd; 48> ... <xyz;1234> ... <абвгд; 21> #####	<i>SD</i>
----------	-----------	-----------	--	-----------

Рисунок 6.6 – Схема хранения небольших каталогов

Элементы списка *RI* упорядочены по именам объектов в алфавитном порядке, что позволяет применить к этому списку *алгоритм бинарного поиска* для определения ссылки на файл по его имени. Бинарный поиск (известный в математике как *метод дихотомии* или *метод деления пополам*) широко применяется в поисковых алгоритмах, так как позволяет отказаться от полного перебора всех *n* элементов списка и сокращает время поиска с $k \times n$ до $k \times \log_2 n$.

Особенно заметным выигрыш в скорости поиска становится при больших длинах списков: если при длине списка $n = 16$ (2^4) бинарный поиск дает только 4-кратное преимущество по сравнению с методом полного сканирования, то при $n = 1024$ (2^{10}) – уже 100-кратное²⁷.

Справедливости ради, следует отметить, что поддержка упорядоченных списков требует дополнительных временных затрат: например, при каждом включении в каталог нового дочернего объекта потребуется повторно выполнять процедуру сортировки элементов списка RI с учетом алфавитной позиции имени добавленного дочернего объекта.

Схема хранения *больших (Large)* каталогов

С увеличением количества дочерних объектов каталога соответственно увеличивается и длина его атрибута RI , и в определенный момент он перестанет помещаться в начальную MFT-запись каталога. В такой ситуации атрибут RI перестает быть *линейным списком* (как это было у небольшого каталога) и получает *иерархическую (древовидную)* структуру, в которой одни элементы списка подчинены другим элементам. При этом часть списка RI («корень дерева») останется в начальной MFT-записи каталога, а для хранения других его частей («листов дерева», подчиненных корню) будут зарезервированы другие MFT-записи, как это показано на рисунке 6.7.

Для преобразования длинного *линейного списка* RI в *дерево*:

- этот список делится на несколько групп элементов с сохранением алфавитного порядка следования элементов;
- каждая такая группа помещается в дополнительную MFT-запись каталога и становится отдельным *листовым* списком, подчиненным *корневому* списку;
- в *корневом* списке остается по одному представителю каждой группы (например, только крайние левые, содержащие «минимальные» по алфавиту имена объектов), а в качестве ссылок используются номера дополнительных MFT-записей, в которых хранятся соответствующие листовые группы.

Поиск подчиненного каталогу объекта по его имени всегда начинается с начальной MFT-записи каталога, в которой хранится *корневая* часть списка RI .

²⁷ Изучение линейных и иерархических структур данных, а также алгоритмов сортировки и поиска данных, реализуемых на таких структурах, выходит за рамки нашего курса.

Бинарным поиском в этом списке определяется ближайшее (по алфавиту) имя группового элемента и далее осуществляется переход по ссылке к соответствующему групповому списку листового уровня. Далее производится бинарный поиск имени искомого объекта в групповом списке, и если объект найден, осуществляется переход по соответствующей ссылке на его начальную MFT-запись.

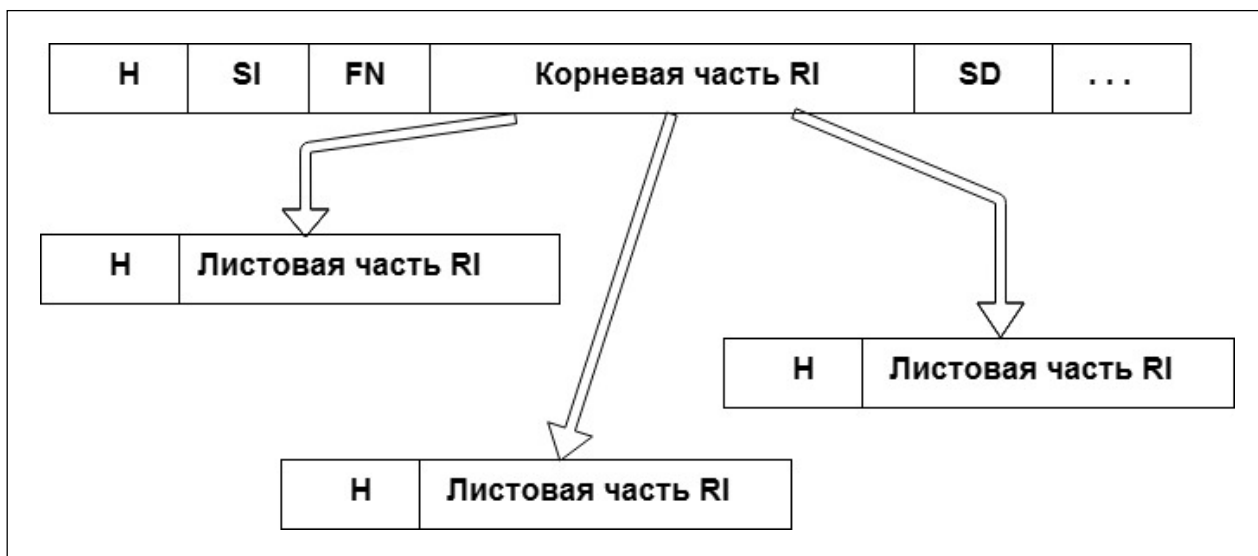


Рисунок 6.7 – Схема хранения больших каталогов

Возможны и другие иерархические схемы хранения каталогов, более сложные по сравнению с рассмотренной двухуровневой схемой. Если количество ссылок в групповом списке превышает допустимый предел, возможно его нерезидентное (в том числе и фрагментарное) хранение в кластерах рабочей зоны тома. При этом в MFT-запись добавляется новый атрибут *IA* (*Index Allocation*), который содержит указатели на фрагменты кластеров (*LN*, *VN*, *CL*) подобно тому, как это сделано в одной из схем хранения больших файлов.

6.3.5 Атрибуты файлов и каталогов

Любой файл NTFS трактуется как набор его атрибутов, в число которых входит и атрибут *DATA*, который может содержать собственно данные файла. Каждый атрибут файла NTFS состоит из полей: *тип* (код) атрибута, *длина* атрибута, *значение* атрибута и, возможно, *имя* атрибута.

Существует два способа хранения атрибутов: резидентное хранение в записях таблицы MFT и нерезидентное хранение – в кластерах рабочей зоны тома. Сортировка файлов (в каталогах) может осуществляться только по их резидентным атрибутам.

Стандартные атрибуты имеют фиксированные коды, имена и форматы хранения, для пользовательских атрибутов все эти параметры не стандартизованы. Обязательный (минимальный) набор стандартных атрибутов приведен в таблице 6.5, их полный перечень – в таблице 6.7.

Таблица 6.7 – Перечень стандартных атрибутов NTFS-файлов

Имя атрибута	Назначение и использование атрибута
<i>Attribute List</i>	Список атрибутов, допустимых для данного файла
<i>File Name</i>	Длинное имя файла, а также номер начальной MFT-записи родительского каталога; если файл содержится в нескольких каталогах (что допустимо в NTFS), то у него будет несколько атрибутов типа <i>File Name</i> ; этот атрибут всегда должен быть резидентным.
<i>MS-DOS Name</i>	Короткое DOS-имя файла в формате 8.3
<i>Version</i>	Номер последней версии файла
<i>Security Descriptor</i>	Информация о защите файла: список прав доступа и поле аудита, которое определяет, какого рода операции над этим файлом нужно регистрировать в журнале транзакций.
<i>Volume Version</i>	Версия тома
<i>Volume Name</i>	Метка тома
<i>Volume Information</i>	Номер версии NTFS
<i>Data</i>	Данные резидентного файла (или ссылки на фрагменты кластеров)
<i>MFT bitmap</i>	Карту использования секторов на томе
<i>Root Index</i>	Корневая часть индексного списка
<i>Index Allocation</i>	Указатели на нерезидентные части индексного списка
<i>External Attribute</i>	Номер первого кластера и количество кластеров нерезидентного атрибута
<i>Standard Information</i>	Вся остальная стандартная информация о файле (например, время создания файла и время последнего обновления файла)

6.4 Контрольные задания

Задание 6.1: Определите понятия «сектор», «дорожка» и «цилиндр», используемые для описания технических характеристик дисковых запоминающих устройств. Какие функции выполняет контроллер дискового устройства ?

Задание 6.2: Каково назначение *главной загрузочной записи (Master Boot Record, MBR)* ? Как используется *MBR* в процессе начальной загрузки ОС ?

Задание 6.3: Какова структура и назначение *таблицы разделов (Partition Table)* ? Как используется *Partition Table* в процессе начальной загрузки ОС и при выполнении файловых операций ?

Задание 6.4: Определите понятие «кластер» как компонент файловой системы. Оцените эффективность использования «мелких» и «крупных» кластеров.

Задание 6.5: Приведите примеры файловых операций, доступных пользователю через интерфейс командной строки. Какие из DOS-команд относятся к категории «внутренних» ?

FAT-системы

Задание 6.6: Перечислите структуры данных, формируемые в системной области FAT-тома. От чего (и как) зависит размер системной области ?

Задание 6.7: Как используется информация загрузочного сектора (*Boot Record*) FAT-тома в процессе начальной загрузки ОС ?

Задание 6.8: Опишите структуру корневого каталога FAT-тома и способ хранения длинных Windows-имен файлов.

Задание 6.9: Опишите структуру байта атрибутов файла.

- какие из атрибутов могут использоваться для обеспечения надежного хранения файлов?
- какие из атрибутов могут использоваться для ограничения доступа к файлам ?
- какой DOS-командой можно просматривать/изменять значения атрибутов файла ?

Задание 6.10: Определите понятие «метка тома».

- где и как хранятся метки FAT-тома ?
- может ли FAT-том иметь несколько меток или не иметь метки?
- какими DOS-командами можно просматривать/изменять значения метки тома ?

Задание 6.11: Опишите структуру FAT-таблицы.

- 6.11.1. От чего (и как) зависит размер FAT-таблицы ?
- 6.11.2. Рассчитайте максимально-допустимое количество кластеров на томе, отформатированном в системах FAT-12, FAT-16 и FAT-32.
- 6.11.3. Рассчитайте минимальный размер кластера на томе емкостью 4 Гб, отформатированном в системе FAT-16.

6.11.4. Рассчитайте максимально возможное количество файлов (с короткими DOS-именами) на FAT-томе размером в 2880 секторов, из которых системная область занимает 43 сектора (в том числе 14 секторов отведено для корневого каталога), а кластеры рабочей области имеют размер по 2 сектора.

Задание 6.12: Опишите алгоритмы выполнения следующих файловых операций:

- 6.12.1. Определение количества «сбойных» кластеров в рабочей области FAT-тома;
- 6.12.2. Определение объема свободного пространства в рабочей области FAT-тома.
- 6.12.3. Поиск файла (по его имени), зарегистрированного в корневом каталоге FAT-тома.
- 6.12.4. Поиск свободных кластеров, необходимых для записи на FAT-том файла заданного размера.
- 6.12.5. Переименование файла.
- 6.12.6. Удаление файла.
- 6.12.7. Восстановление удаленного файла.
- 6.12.8. Создание подчиненного каталога.

Задание 6.13: * Разработайте программную модель FAT-таблицы, реализующую следующие алгоритмы:

- 6.13.1. Определение количества «сбойных» кластеров в рабочей области FAT-тома;
- 6.13.2. Определение объема свободного пространства в рабочей области FAT-тома.
- 6.13.3. Определение общего количества файлов всех типов на томе.
- 6.13.4. Поиск свободных кластеров, необходимых для записи на FAT-том файла заданного размера.

Задание 6.14: * Разработайте программную модель каталога, реализующую следующие алгоритмы:

- 6.14.1. Поиск в каталоге файла по его имени.
- 6.14.2. Переименование файла.
- 6.14.3. Поиск в каталоге всех файлов, размер которых превышает заданное значение.
- 6.14.4. Поиск в каталоге файла по номеру его начального кластера

6.14.5. Поиск в каталоге записей обо всех удаленных файлах.

6.14.6. Поиск в каталоге всех подчиненных ему каталогов.

Задание 6.15: * На основе программных моделей каталога и FAT-таблицы разработайте программы, реализующие следующие алгоритмы:

6.15.1. Получение списка номеров кластеров, занятых файлом каталога.

6.15.2. Получение списка имен файлов каталога, размещенных НЕ-фрагментарно.

6.15.3. Удаление файла.

6.15.4. Восстановление удаленного файла.

6.15.5. Создание подчиненного каталога.

Задание 6.16: * Разработайте программу, реализующую алгоритм формирования короткого DOS-имени файла из его длинному Windows-имени.

Задание 6.17: Перечислите основные недостатки FAT-систем и возможные причины проявления этих недостатков.

NTFS-системы

Задание 6.18: Прокомментируйте понятия «файл» и «атрибут файла» в NTFS. Как взаимосвязаны эти понятия ?

Задание 6.19: Перечислите стандартные атрибуты файлов. Какие из них являются обязательными для файлов ?

Задание 6.20: Какую роль в файловой системе выполняет MFT-файл ? Как ограничен размер этого файла ?

Задание 6.21: Какие файлы называют резидентными ? Сколько кластеров занимает резидентный файл в рабочей зоне NTFS-тома? Как организован доступ к резидентным файлам ?

Задание 6.22: Как организовано хранение больших по размеру и сильно фрагментированных файлов?

Задание 6.23: Как организован в NTFS-системах поиск файлов по значениям их атрибутов ?

ОБМЕН ДАННЫМИ С ПЕРИФЕРИЙНЫМИ УСТРОЙСТВАМИ**7.1 Система обработки прерываний**

Понятием «прерывание» обозначают механизм, обеспечивающий оперативную реакцию центрального процессора на определенное событие, требующее прерывания выполнения исполняемой процессором программы и перехода к выполнению другой специальной программы, называемой *обработчиком прерывания*. После завершения процесса обработки прерывания автоматически происходит возврат к выполнению прерванной программы.

Система обработки прерываний способна обрабатывать до 256 различных событий, все они пронумерованы в диапазоне от INT 00_h до INT FF_h, и с каждым номером прерывания связана определенная программа²⁸. Ряд номеров прерываний из этого диапазона зарезервированы системой, все остальные могут использоваться прикладными программами, в том числе и для обработки событий, связанных с обслуживанием нестандартного периферийного оборудования, подключаемого к компьютеру.

Система обработки прерываний (как, впрочем, и любая другая программная система) состоит из трех взаимодействующих компонентов:

- аппаратного комплекса, основу которого составляет *контроллер прерываний*, обеспечивающий взаимодействие периферийных устройств с центральным процессором;
- программного обеспечения, представленного множеством *программ обработки прерываний*;
- структур данных, главная из которых – это *таблица векторов прерываний*, организуемая в ОЗУ и выполняющая роль адресного справочника, используемого для хранения указателей на программы обработки прерываний.

По типу источников событий требующих оперативной реакции процессора в режиме прерываний, различают *аппаратные* и *программные* прерывания, аппаратные прерывания, в свою очередь, могут быть *внутренними* или *внешними*.

²⁸ Не все номера из этого диапазона являются номерами *прерываний* – ряд номеров (например, INT 1F_h и INT 43_h) используются системой для организации доступа к служебным структурам данных, размещенных в памяти ПК.

7.1.1 Аппаратные прерывания

Инициатором *внутренних аппаратных прерываний* может быть сам центральный процессор, если он обнаружит нештатную ситуацию при выполнении очередной машинной команды. Например, событие «деление на ноль» вызовет прерывание INT 00_h, а прерывание INT 04_h зарезервировано за событием «переполнение» при выполнении арифметической операции.

Запросы на обработку внутренних аппаратных прерываний генерируются внутренними блоками процессора, а индикаторами событий, вызывающих такие прерывания, могут быть соответствующие биты *регистра флагов* центрального процессора (п. 4.3), содержащего, в частности, коды результатов выполненных команд.

Внешние аппаратные прерывания обеспечивают асинхронное взаимодействие центрального процессора с периферийным оборудованием, подключаемым к системной магистрали через соответствующие адаптеры (рисунок 4.2), и являются следствием наступления некоторого внешнего события, например: прочитан очередной сектор диска, нажата (или отпущена) клавиша клавиатуры, принтер послал сигнал готовности, или сработал датчик внешнего оборудования, управляемого компьютером.

Адаптеры периферийных устройств передают запросы *контроллеру прерываний* (п. 7.1.3), основная задача которого – это идентификации устройства, инициировавшего такой запрос, и формирование соответствующего *номера прерывания*. Контроллер прерываний выставляет этот номер на системную шину данных, а по шине управления передает центральному процессору специальное сообщение – запрос на прерывание.

Центральный процессор, получив запрос на прерывание, читает с шины данных номер прерывания и запускает стандартную *процедуру обработки прерывания*, алгоритм реализации которой рассмотрен в п. 7.1.5.

7.1.2 Программные прерывания

Источниками программных прерываний, как это следует из их названия, являются программы – как системные, так и прикладные. Процессор, обнаружив в машинном коде команду *INT n*, запускает стандартную процедуру обработки прерывания (п. 7.1.5), которая вызывает программу обработки прерывания № *n*, которая к этому моменту должна быть загружена в память ПК.

Программные прерывания эффективно используются в MS DOS для обращения к множеству системных функций: например, прерывание INT10_h обеспечивает доступ к системным функциям видео-сервиса, прерывание INT 16_h – к функциям, обслуживающих ввод данных с клавиатуры, а прерывание INT 21_h – к функциям управления памятью и к функциям, реализующим файловые операции, рассмотренные в главе 6 учебного пособия.

Программное прерывание не является «неожиданным» для центрального процессора и синхронизировано с процессом выполнения команд машинной программы – в этом смысле оно является «искусственным», использующим специфический способ запуска программ, созданный для обработки аппаратных прерываний. Для запуска такой программы достаточно связать с ней номер некоторого прерывания и записать соответствующий указатель в *таблицу векторов прерываний* (п. 7.1.4).

7.1.3 Контроллер прерываний

Контроллер прерываний – специализированная микросхема (или несколько каскадно-соединенных микросхем), к входным линиям которой подключаются контроллеры периферийных устройств – источников прерываний. В качестве источников прерываний могут выступать контроллеры клавиатуры, системного таймера или датчиков различных технических устройств, требующих реакции центрального процессора на поступающие от них сигналы.

Существенным здесь является то, что центральный процессор не может прогнозировать время поступления этих сигналов – согласитесь, что трудно предугадать момент времени, когда задумчивый пользователь ПК найдет на клавиатуре нужную ему клавишу и, наконец, нажмет на нее, и еще труднее прогнозировать моменты срабатывания подключенных к компьютеру датчика задымленности воздуха в помещении или датчика температуры тормозной колодки автомобиля, управляемого бортовым компьютером. При этом процессор должен обеспечить оперативную (и правильную) реакцию на каждый из поступивших сигналов путем запуска соответствующих программ, временно прерывая выполнение других программ, некоторые команды которых к этому моменту уже были загружены в регистр очереди команд (рисунок 4.3).

На рисунке 7.1 приведена упрощенная функциональная схема контроллера прерываний Intel 8259A, используемого в IBM-совместимых ПК. Этот контроллер представлен в адресном пространстве ввода-вывода двумя портами (20_h и

21h), через которые производится обмен данными и управляющими сигналами с центральным процессором ПК.

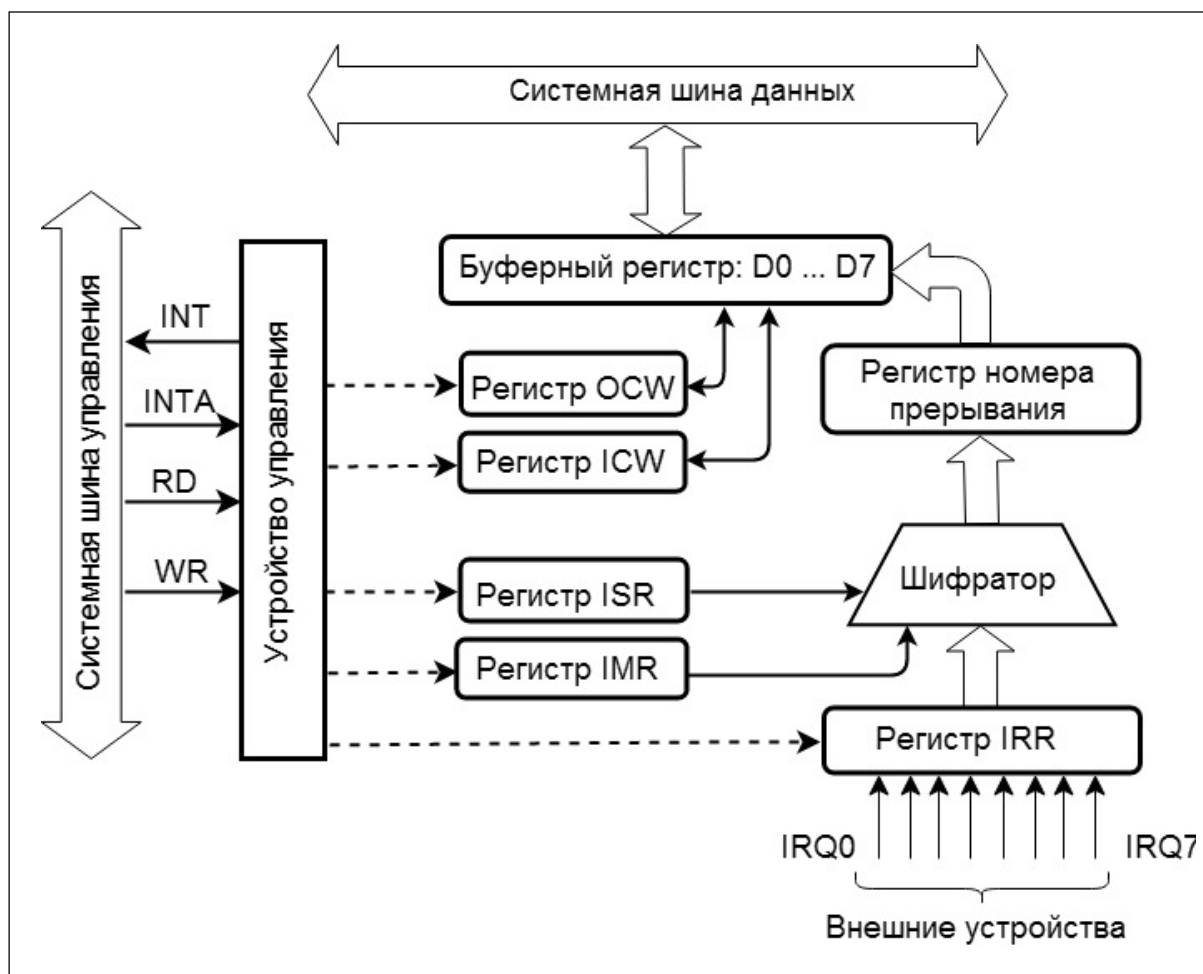


Рисунок 7.1 – Функциональная структура контроллера прерываний

Intel 8259A – это программируемый контроллер, который может находиться в двух состояниях: в рабочем состоянии обслуживания запросов и в состоянии настройки. Основная задача, решаемая контроллером в состоянии обслуживания запросов, – идентификация внешних событий, требующих прерывания программы, выполняемой центральным процессором.

Обобщенный алгоритм решения этой задачи можно представить следующим набором шагов:

- прием запроса на прерывание от внешнего устройства;
- идентификация источника запроса;
- формирование номера прерывания, соответствующего источнику;
- передача номера прерывания центральному процессору;
- передача центральному процессору сообщения о прерывании;

- прием от центрального процессора подтверждения приема переданного сообщения.

Приведем более детальное описание этого алгоритма и прокомментируем назначение основных функциональных блоков контроллера, показанных на рисунке 7.1.

Контроллер ожидает приема запросов на прерывания – сигналов IRQ0 ... IRQ7²⁹ от контроллеров внешних устройств, и временно сохраняет принятые запросы в 8-разрядном регистре *IRR (InterRupt Register)*. Единичное значение какого-либо бита этого регистра означает поступление запроса от устройства, подключенного к соответствующей линии IRQ.

Номер входной линии IRQ определяет *уровень прерывания*, устанавливающий приоритет запроса: как правило (если задан режим фиксированных приоритетов), наивысший приоритет имеет линия IRQ0, и далее в порядке убывания – до линии IRQ7, имеющей низший приоритет.

Восьмиразрядный *регистр маскирования прерываний IMR (Interrupt Mask Register)* используется для запрета (маскирования) обработки запросов, поступающих на соответствующие входы IRQ: единичное значение *i*-го бита регистра *IMR* маскирует запросы прерываний, поступающие на *i*-й IRQ-вход регистра *IRR*. Программирование регистра *IMR* производится через порт 21h.

Единичное значение *i*-го бита восьмиразрядного *регистра обслуживаемых прерываний ISR (Interrupt Service Register)* показывает, прерывания каких уровней обрабатываются в микропроцессоре.

Устройство управления контроллера вырабатывает выходной сигнал INT, связанный по системной шине управления с входом INTR микропроцессора (рисунки 4.3), и ожидает поступления от микропроцессора сигнала INTA, подтверждающего факт принятия им прерывания на обслуживание.

Микропроцессор, получив по шине управления сигнал INTR, анализирует состояние флага IF (рисунки 4.3), единичное состояние которого разрешает, а нулевое – запрещает аппаратные прерывания.

Если прерывания запрещены (IF = 0):

- прерывание не выполняется до момента установки IF в единицу;

Иначе (IF = 1):

²⁹ Предусмотрена возможность увеличения числа линий запросов путем каскадного подключения еще одного контроллера прерываний. В этом случае линия IRQ2 не используется для подключения внешнего устройства, она обеспечивает связь между основным и «ведомым» контроллерами прерываний.

- флаг IF сбрасывается в ноль;
- запускается процедура обработки прерывания, по завершению которой флаг IF устанавливается в единицу;
- формируется выходной сигнал подтверждения прерывания INTA, связанный по системной шине управления с одноименным входным сигналом контроллера прерывания.

Контроллер прерываний, получив по системной шине управления сигнал подтверждения прерывания INTA:

- сбрасывает в ноль бит регистра IRR, соответствующий уровню IRQ обработанного процессором прерывания;
- устанавливает в единицу соответствующий бит регистра ISR, что фиксирует факт принятия прерывания к обработке процессором;
- в соответствии с состоянием регистров *IRR*, *IMR* и *ISR*, формирует *номер прерывания* – однобайтовое число, соответствующее уровню прерывания, то есть номеру активной линии IRQ. Номер прерывания – это, по существу, числовое «имя» внешнего события, требующего прерывания работы процессора. Например, контроллер системного таймера соединен с линией высшего приоритета IRQ0, и ему стандартно присвоен 8-й номер, а контроллер клавиатуры связан с линией IRQ1, и этой линии соответствует прерывание №9.
- Шифратор выполняет также функцию *арбитра приоритетов*, разрешающего конфликты при одновременном поступлении IRQ-запросов разных уровней. Если поступившее прерывание уровня *i* не маскировано ($IMR[i] = 0$), и приоритет прерывания, обрабатываемого процессором в данный момент, ниже приоритета поступившего прерывания ($ISR > i$), шифратор вычисляет *номер прерывания*, который записывается в регистр номера прерывания и далее, через *буферный регистр*, передается на системную шину данных, доступную центральному процессору. В микропроцессоре этот номер используется для вычисления адреса программы обработки прерывания.

Если до завершения обработки прерывания поступит очередной IRQ-запрос, то его приоритет определит дальнейшие действия контроллера:

- Если приоритет вновь поступившего IRQ-запроса ниже или равен приоритету выполняемого прерывания:
 - новый IRQ-запрос будет запомнен в регистре IRR установкой в единицу соответствующего бита;

- обслуживание этого прерывания будет отложено.
- Если приоритет вновь поступившего IRQ-запроса выше приоритета выполняемого прерывания:
 - прерывается процедура обработки текущего прерывания;
 - вызывается процедура обработки нового прерывания.

В состоянии настройки контроллер принимает и сохраняет в одноименных регистрах команды инициализации *ICW* (*Initialization Command Words*), задающие режим его работы, и операционные команды *OCW* (*Operation Control Words*), позволяющие перепрограммировать контроллер, например, маскировать или размаскировать прерывания, менять их уровни (приоритеты).

Соответствующим программированием контроллера можно устанавливать специальные режимы завершения прерываний или, например, переводить контроллер в режим опроса (*Polling Mode*), в котором аппаратные прерывания не происходят автоматически, а центральный процессор считывает содержимое регистра *IRR*, получает от контроллера информацию о наличии и уровнях поступивших запросов на прерывания и сам принимает решение о запуске процедуры прерывания.

7.1.4 Таблица векторов прерываний

Вектором прерывания называется начальный адрес программы обработки этого прерывания (или адрес иной структуры данных, связанной с этим прерыванием). Вектор прерывания хранится в сегментной форме (п. 4.6.1) и занимает в памяти два двухбайтовых машинных слова: слово со старшим адресом содержит номер сегмента, а слово с младшим адресом – смещение от начала этого сегмента.

Хранение векторов прерываний организовано в специальной структуре данных, называемой *таблицей векторов прерываний*, которая позволяет однозначно связать *номера прерываний* с соответствующими им *векторами прерываний*. Таблица векторов прерываний формируется в ОЗУ и занимает первый килобайт нулевого сегмента адресного пространства (от $[0000:0000]_h$ до $[0000:3FFF]_h$). Все векторы прерываний расположены в этой «таблице» линейно в порядке возрастания соответствующих им номеров прерываний (от № 00_h до № FF_h), и каждый вектор занимает в таблице 4 байта, что позволяет использовать простую формулу $4 \times n$ для определения смещения вектора *n*-ного прерывания относительно начала таблицы.

Таким образом, *номер прерывания n*, умноженный на 4, определяет сегментный адрес 4-байтовой ячейки памяти $[0000:4 \times n]_h$, в которой расположен

вектор n -ного прерывания, который, сам являясь сегментным адресом, выполняет роль указателя на программу обработки этого (n -ного) прерывания.

На рисунке 7.2 показан начальный фрагмент таблицы векторов прерываний. По адресу $[0000:0000]_h$ хранится сегментный адрес $[00A7:1068]_h$ – это вектор 0-го прерывания, представленный сегментом № $[00A7]_h$ и смещением $[1068]_h$ от начала этого сегмента. Следующие 4 байта представляют вектор 1-го прерывания $[0070:018B]_h$, затем – 2-го, и т.д. до последнего вектора прерывания с самым большим номером FF_h .

0000:0000	68 10 A7 00 8B 01 70 00	16 00 96 03 8B 01 70 00
0000:0010	8B 01 70 00 B9 06 0C 02	40 07 0C 02 FF 03 0C 02
0000:0020	83 01 EC 0F EC 06 11 0E	3A 00 96 03 54 00 96 03
0000:0030	6E 00 96 03 88 00 96 03	A2 00 96 03 FF 03 0C 02
0000:0040	A9 08 0C 02 A4 09 0C 02	AA 09 0C 02 5D 04 0C 02
0000:0050	B0 09 0C 02 0D 02 DB 02	C4 09 0C 02 8B 05 0C 02
0000:0060	0E 0C 0C 02 14 0C 0C 02	1F 0C 0C 02 AD 06 0C 02
0000:0070	AD 06 0C 02 A4 F0 00 F0	37 05 0C 02 F1 8F 00 C0
0000:0080	72 10 A7 00 7C 10 A7 00	7F 01 15 04 4B 01 15 04
0000:0090	56 01 15 04 86 10 A7 00	90 10 A7 00 9A 10 A7 00
0000:00A0	C5 01 EC 0F 54 02 70 00	F2 04 9A D3 B8 10 A7 00
0000:00B0	B8 10 A7 00 B8 10 A7 00	40 01 15 04 10 0E 11 0E
0000:00C0	EA AE 10 A7 00 38 00 F0	B8 10 A7 00 C4 23 02 D0
0000:00D0	B8 10 A7 00 B8 10 A7 00	B8 10 A7 00 B8 10 A7 00
0000:00E0	B8 10 A7 00 B8 10 A7 00	B8 10 A7 00 B8 10 A7 00
0000:00F0	B8 10 A7 00 B8 10 A7 00	B8 10 A7 00 B8 10 A7 00
0000:0100	8A 04 0C 02 FF 03 0C 02	76 09 0C 02 F1 A1 00 C0
0000:0110	F3 38 00 F0 F3 38 00 F0	C0 00 40 00 F3 38 00 F0
0000:0120	F3 38 00 F0 F3 38 00 F0	F3 38 00 F0 F3 38 00 F0
0000:0130	F3 38 00 F0 F3 38 00 F0	F3 38 00 F0 F3 38 00 F0
0000:0140	F3 38 00 F0 F3 38 00 F0	01 00 00 00 F3 38 0F F0
0000:0150	F3 38 00 F0 F3 38 00 F0	F3 38 00 F0 F3 38 00 F0

Рисунок 7.2 – Фрагмент таблицы векторов прерываний

Таблица векторов прерываний частично инициализируется BIOS перед началом загрузки DOS, частично – при загрузке DOS. Пользовательские программы также могут модифицировать эту таблицу, используя номера свободных прерываний или переустанавливая некоторые из уже используемых векторов прерываний.

7.1.5 Процедура обработки прерывания

Если запрос прерывания принят на обслуживание центральным процессором, выполняется следующая типовая процедура:

1) Перед тем, как исполняемая процессором программа будет прервана, все ее оперативные данные (промежуточные результаты, временно сохраненные в регистрах общего назначения, содержимое регистра флагов и буфера очереди команд) сохраняются в области оперативной памяти, отведенной под стек. По-

ложение стека в ОЗУ определяется состоянием соответствующих адресных регистров процессора (рисунок 4.3): номер сегмента стековой памяти определяет сегментный регистр SS (*Stack Segment*), а вершину стека – регистр-указатель SP (*Stack Pointer*). Вершина стека – это единственная его точка, через которую происходит как запись данных в стек, так и извлечение из стека записанных в него данных, что позволяет реализовать протокол *LIFO* обслуживания стека (*Last Input – First Output*).

2) С системной шины данных считывается номер прерывания n , переданный на эту шину контроллером прерываний.

3) Формируется сегментный адрес $[0000:4 \times n]_h$ точки входа в таблицу векторов прерываний, и считывается вектор прерывания, расположенный по этому адресу.

4) Выполняется программа обработки прерывания, на которую «указывает» вектор прерывания:

- компоненты вектора прерывания (номер сегмента и смещение) помещаются в адресные регистры процессора CS и IP;
- данные этих регистров обрабатываются сумматором адреса, формирующим начальный (линейный) адрес программы обработки прерывания;
- по адресной шине запрашивается и затем выполняется первая команда этой программы, и так далее до тех пор, пока не будет выполнена команда «конец программы».

5) По завершению программы обработки прерывания из стека восстанавливаются данные прерванной программы, и продолжается выполнение ее очередной команды.

Сохранение в стековой памяти данных прерванной программы обеспечивает возможность иерархической обработки прерываний. Это означает, что если в процессе обработки одного прерывания процессору поступает запрос на выполнение прерывания более высокого приоритета, первый обработчик прерывания будет прерван (то есть теперь уже к нему будет применена рассмотренная выше процедура) и продолжит свое выполнение только после завершения работы программы обработки второго (более приоритетного) прерывания.

Управление приоритетами внешних аппаратных прерываний осуществляет контроллер прерываний (рисунок 7.1), принимающий запросы от адаптеров периферийных устройств на свои входные линии IRQ: с каждой линией связан

определенный уровень приоритета. Если до окончания обработки высокоприоритетного прерывания поступает запрос низкого приоритета, он становится в очередь (в соответствии со своим приоритетом), в противном случае запрос передается центральному процессору для исполнения.

7.2 Клавиатура

Клавиатура персонального компьютера – одно из важнейших его периферийных устройств, предназначенное для ввода текстовой (буквенно-цифровой) и управляющей информации.

Клавиатура является программно-управляемым устройством и имеет в своем составе специализированный контроллер, основной функцией которого является отслеживание фактов нажатия и отпускания клавиш путем циклического сканирования наборного поля клавиатуры, в результате которого формируются *скан-коды* нажатых (или отпущенных) клавиш.

Скан-код – это однобайтовый номер, присваиваемый каждой клавише. Семь младших битов скан-кода – это собственно уникальный код клавиши, а старший бит используется для кодирования факта её нажатия (0) или отпускания (1). При такой системе кодирования с каждой клавишей связано два ее аскан-кода: код нажатой клавиши всегда на 128 (2^7) меньше, чем код этой же клавиши, формируемый при ее отпускании, а общее количество кодируемых клавиш ограничивается числом 128: коды в диапазоне $[0 \dots 7F_{\text{h}}]$ соответствуют нажатым клавишам, а коды в диапазоне $[80_{\text{h}} \dots FF_{\text{h}}]$ – отпущенным. Заметим, что такое ограничение не противоречит требованиям стандартных форматов клавиатур, предусматривающих 101, 104, 108 или 109 клавиш.

Следует понимать, что скан-коды клавиш определяются «схемой раскладки» наборного поля клавиатуры и напрямую не связаны с обозначениями символов, нанесенными на их поверхность. Размещение символов на наборном поле клавиатуры (так называемая *раскладка клавиатуры*) определено соответствующими стандартами, а соответствие между скан-кодом клавиши и ASCII-кодом одного из связанных с ней символов определяется программно (п. 7.2.2) в соответствии с используемой кодовой таблицей символов.

7.2.1 Контроллер клавиатуры

Процесс ввода данных с клавиатуры реализуется тремя аппаратными устройствами компьютера – контроллером клавиатуры, контроллером прерываний и центральным процессором.

Контроллер клавиатуры взаимодействует с центральным процессором через систему прерываний (п.7.1) и связан с линией IRQ-1 контроллера прерываний, что обеспечивает клавиатуре высокий уровень приоритета (2-й после приоритета системного таймера, связанного с линией IRQ-0).

Контроллер клавиатуры – это однокристалльная микро-ЭВМ со своим «центральным» процессором, ПЗУ, устройствами ввода и вывода данных.

В качестве устройства ввода данных используется *наборное поле* клавиатуры – электронно-механическое (или иное) устройство, схематично показанное на рисунке 7.3. Электрические линии строк и столбцов образуют прямоугольную матрицу, в узлах которой расположены нормально разомкнутые контакты, управляемые соответствующими клавишами. Таким образом, каждая клавиша клавиатуры компьютера однозначно идентифицируется номером $[i,j]$ связанного с ней узла матрицы наборного поля.

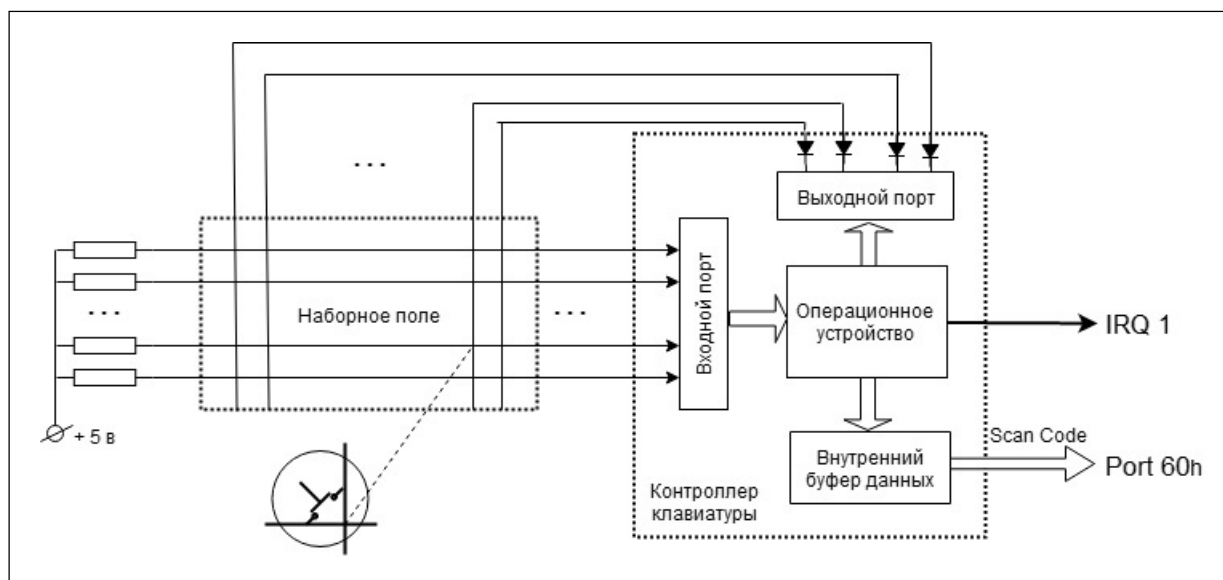


Рисунок 7.3 – Схема наборного поля клавиатуры

Устройство ввода данных контроллера клавиатуры представлено двумя внутренними портами: входной порт, связанный с линиями строк матрицы наборного поля, подключенными через резисторы к источнику питания, уровень напряжения которого (+5 в) соответствует логической единице, и выходной порт, подающий сигналы уровня логического нуля на линии столбцов.

При отпущенных клавишах контакты наборного поля разомкнуты, и сигналы всех линий строк, поступающие на входной порт контроллера, соответствуют логической единице, то есть каждая $[i,j]$ -тая клавиша находится в состоянии $[0;1]$.

При нажатии клавиши соответствующий $[i,j]$ -тый контакт наборного поля замыкается, и в очередном цикле сканирования в момент подачи нулевого сигнала на i -тый столбец входной сигнал с j -той строки получит нулевое значение. В результате $[i,j]$ -тая клавиша сменит состояние с $[0;1]$ на $[0;0]$, что является индикатором события « $[i,j]$ -тая клавиша *нажата*».

При отпуске нажатой клавиши ее состояние изменится с $[0;0]$ на $[0;1]$, этот факт будет обнаружен операционным устройством контроллера в следующем цикле сканирования наборного поля и использован для регистрации противоположного события « $[i,j]$ -тая клавиша *отпущена*».

Программа, хранимая в ПЗУ контроллера и выполняемая в его операционном устройстве, реализует следующий алгоритм в каждом цикле сканирования наборного поля (с периодичностью порядка 10 раз в секунду):

- на каждую i -тую линию столбцов матрицы через выходной порт последовательно посылается электрический сигнал, уровень напряжения которого соответствует логическому нулю;
- считываются сигналы, поступающие на входной порт от каждой j -той линии строк.
- контролируется изменение состояния каждой $[i,j]$ -той клавиши;
- клавише, сменившей свое состояние, программно присваивается уникальный для нее 7-битный код и к этому коду добавляется старший бит (0 – при нажатии клавиши, 1 – при отпуске) – в результате формируется скан-код клавиши;
- скан-код клавиши помещается во внутренний буфер данных, выполняющий роль устройства вывода контроллера клавиатуры;
- из внутреннего буфера данных скан-код нажатой (или отпущенной) клавиши передается в порт 60_h адресного пространства ввода-вывода, доступный программам, выполняемым центральным процессором;
- на линию IRQ1 контроллера прерываний посылается сигнал запроса прерывания.

При удержании клавиши в нажатом состоянии, контроллер клавиатуры переходит в режим автоповтора, в котором периодически генерируется скан-код нажатой клавиши.

Далее, уже без участия контроллера клавиатуры, выполняется процедура обработки прерывания, детально описанная выше (п.4.5, п.7.1.4 и п.7.1.5):

- контроллер прерывания принимает запрос по линии IRQ1, формирует соответствующий этой линии номер прерывания «9», выставляет этот номер на шину данных и посылает центральному процессору запрос на прерывание по шине управления;
- центральный процессор прерывает выполняемую программу, определяет вектор прерывания, соответствующий полученному по шине данных номеру прерывания, и запускает программу обработки 9-го прерывания, соответствующую этому вектору.

7.2.2 Алгоритм обработки прерывания клавиатуры

Программист, написавший в исходном коде прикладной программы `input()`, `cin`, `getchar()`, `readln()` или `readkey`, надеется на транслятор, который должен успешно справиться с задачей получения символьной информации от стандартного устройства ввода. У разработчика транслятора также есть надежда на то, что вызов соответствующей системной функции избавит его от необходимости беспокоиться о деталях низкоуровневой реализации процесса обмена данными с клавиатурой. На чем же основаны эти надежды прикладных и системных программистов? В том числе, и на доверии к обработчику прерывания №9, обеспечивающему программный интерфейс между контроллером клавиатуры и прикладными программами, ожидающими получения видимых пользователями символьных данных.

На рисунке 7.4 приведена общая блок-схема алгоритма, реализуемого программой обработки 9-го прерывания. Основное назначение этой программы – преобразование принятого скан-кода клавиши в код символа и передача этого кода прикладной программе для последующей обработки.

Кроме этого, обработчик 9-го прерывания генерирует и сохраняет так называемые *расширенные скан-коды* для вспомогательных клавиш или основных буквенно-цифровых клавиш при их нажатии в комбинациях с управляющими клавишами *Ctrl* и/или *Alt*, а также классифицирует принятые скан-коды управляющих клавиш и клавиш-переключателей, сохраняя информацию о текущем статусе клавиатуры для последующего использования как прикладными программами, так и самим обработчиком, например, для корректного преобразования скан-кодов клавиш в ASCII-коды символов.

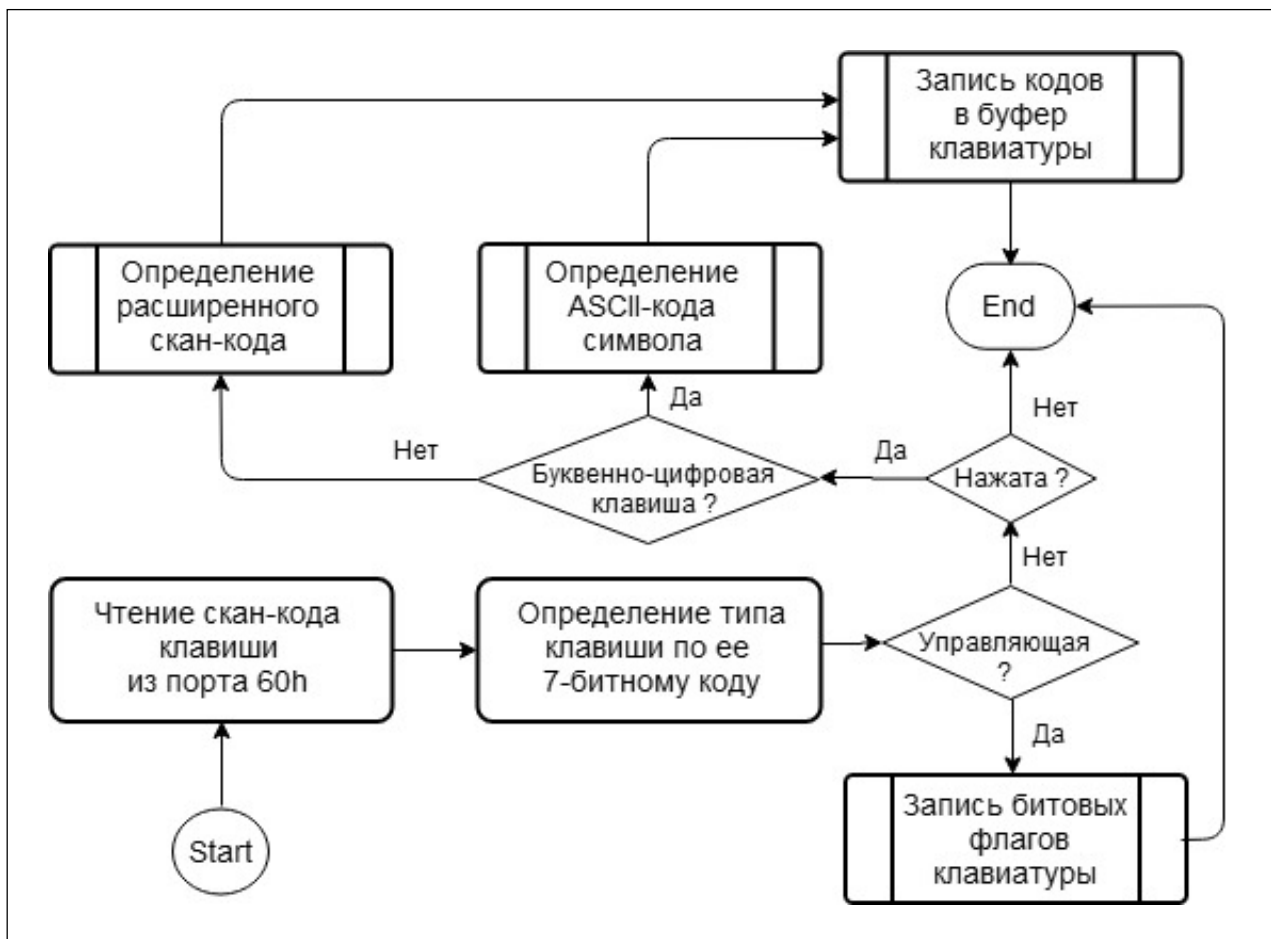


Рисунок 7.4 – Алгоритм обработки прерывания №9

7.2.2.1 Структуры данных, используемые обработчиком 9-го прерывания

Программа обработки прерывания №9 использует три структуры данных, зарезервированных для нее в области переменных BIOS (таблица 7.1):

- два байта битовых «флагов» – индикаторов состояния управляющих клавишей и клавишей-переключателей;
- один байт для накопления ASCII-кода символа при его прямом вводе (alt+цифра);
- буфер клавиатуры, используемый для передачи прикладной программе ASCII-кода введенного пользователем символа или расширенного скан-кода вспомогательной клавиши или комбинации клавишей.

Таблица 7.1 - Блок данных BIOS, обслуживающий ввод с клавиатуры

Начальный адрес	Длина, байтов	Назначение	
[0040:0017] _h	2	Битовые флаги (статусы) клавиатуры – индикаторы состояния управляющих клавиш (рисунок 7.4)	
[0040:0019] _h	1	Текущее (накопленное) значение вводимого ASCII-кода символа (Alt + цифра)	
[0040:001A] _h	2	Буфер клавиатуры	Указатель на «хвост» буфера – адрес (смещение по 40-му сегменту) ячейки, в которую обработчик 9-го прерывания должен записать код последнего символа, введенного с клавиатуры.
[0040:001C] _h	2		Указатель на «голову» буфера – адрес ячейки, из которой прикладная программа должна прочесть код очередного введенного, но еще не прочитанного символа.
[0040:001E] _h	Обычно 32		Собственно буфер (16 двухбайтовых элементов)
[0040:0080] _h	2		Начальный адрес буфера (обычно – 001E _h)
[0040:0082] _h	2		Конечный адрес буфера +1 (обычно – 003E _h)

7.2.2.2 Формирование «флагов» клавиатуры

Как показано на рисунке 7.4, обработчик 9-го прерывания начинает работу с чтения содержимого порта 60_h и анализа прочитанного скан-кода последней нажатой (или отпущенной) клавиши.

Если прочитанный скан-код соответствует одной из управляющих клавиш *Ctrl*, *Alt*, *Shift*, или одной из клавиш-переключателей *CapsLock*, *NumLock*, *ScrollLock*, *SysReq*, *Pause* или *Insert*, программа устанавливает соответствующие биты флагов клавиатуры (таблица 7.2) в «1» или «0» в зависимости от значения старшего бита скан-кода клавиши (нажата или отпущена), а для клавиш-переключателей – дополнительно изменяет значения соответствующих битов на противоположное.

Например, при удержании любой из двух клавиш *Ctrl* второй бит первого байта флагов будет установлен в «1», а при нажатой правой клавише *Ctrl* дополнительно устанавливается в «1» первый бит второго байта флагов.

Еще один пример: при нажатии клавиши *CapsLock* 6-й бит первого и второго байтов флагов устанавливается в «1», что соответствует ситуации: «установлен верхний регистр ввода символов, и клавиша *CapsLock* удерживается нажатой».

При отпускании этой клавиши 6-й бит второго байта будет сброшен в «0», а первый байт останется без изменений, что соответствует ситуации «установлен верхний регистр ввода символов, и клавиша *CapsLock* отпущена». После повторного нажатия и отпускания клавиши *CapsLock* оба этих бита будут сброшены в «0», что соответствует ситуации «установлен нижний регистр ввода символов, и клавиша *CapsLock* отпущена».

Таблица 7.2 – Структура байтов флагов клавиатуры

1-й байт флагов [0040:0017] _h		2-й байт флагов [0040:0018] _h	
Бит	Статус клавиатуры при единичном значении бита	Бит	Статус клавиатуры при единичном значении бита
0	Нажата правая клавиша <i>Shift</i>	0	Нажата правая клавиша <i>Ctrl</i>
1	Нажата левая клавиша <i>Shift</i>	1	Нажата левая клавиша <i>Alt</i>
2	Нажата (любая) клавиша <i>Ctrl</i>	2	Нажата клавиша <i>SysReq</i>
3	Нажата (любая) клавиша <i>Alt</i>	3	Установлен режим «Pause»
4	Установлен режим «ScrollLock»	4	Нажата клавиша <i>ScrollLock</i>
5	Установлен режим «NumLock»	5	Нажата клавиша <i>NumLock</i>
6	Установлен режим «CapsLock»	6	Нажата клавиша <i>CapsLock</i>
7	Установлен режим «Insert»	7	Нажата клавиша <i>Insert</i>

Анализ структуры байтов флагов клавиатуры (таблица 7.2) позволяет сделать следующие выводы:

1) Парные управляющие клавиши *Shift*, *Ctrl* и *Alt* (отличающиеся значением своих скан-кодов) имеют отдельные позиции в байтах флагов, что позволяет идентифицировать ситуации с удержанием как левых или правых одноименных клавиш, так и обеих клавиш одновременно.

2) Информация о клавишах-переключателях позволяет не только идентифицировать установленный режим работы клавиатуры, но и определить текущее состояние клавиши-переключателя, установившей этот режим.

7.2.2.3 Алгоритм вычисления кодов символов

Все клавиши, не отнесенные к категориям управляющих и клавишей-переключателей (таблица 7.2), формально относятся к категории символьных клавишей. В эту категорию попадают все буквенно-цифровые клавиши, а также вспомогательные клавиши, например, функциональные F1 ... F12, клавиши табуляции и управления курсором, клавиши Delete и BackSpace, которые фактически символьными не являются и не требуют трансляции своих скан-кодов в ASCII-коды каких-либо символов. Объединяет все эти «формально символьные» клавиши только то, что информация об их нажатии хранится в буфере клавиатуры в соответствующем двухбайтовом коде.

Если принятый из порта 60h скан-код соответствует «формально символьной» клавише, и при этом старший бит скан-кода этой клавиши имеет значение «0» (клавиша нажата), выполняется блок алгоритма (рисунок 7.4), обеспечивающий трансляцию скан-кода либо в ASCII-код символа (для буквенно-цифровой клавиши), либо в расширенный скан-код (для вспомогательной клавиши или комбинации нескольких «одновременно» нажатых клавишей).

Основные буквенно-цифровые клавиши – ASCII-код символа.

Транслятор скан-кода буквенно-цифровой клавиши в ASCII-код соответствующего ей символа может быть построен на базе бинарной таблицы, каждая строка которой представляет одну из таких клавиш в соответствии со стандартной раскладкой клавиатуры и используемой кодовой таблицей символов: в левом столбце трансляционной таблицы записан скан-код клавиши, а в правом – ASCII-код символа. Такая структура таблицы позволит найти код символа по заданному значению скан-кода, но при этом потребуются решить проблему многозначности: одной клавише может соответствовать, как минимум, два кода символа в зависимости от установленного регистра ввода.

Универсальное решение этой проблемы – сделать бинарную трансляционную таблицу тернарной, добавив третий столбец с ASCII-кодами заглавных букв и с кодами «верхних» символов цифровых клавишей. Выбор одного из двух ASCII-столбцов в процессе трансляции зависит от состояния соответствующих флаговых битов (таблица 7.2), связанных с клавишами *Shift* и *CapsLock*. Возможна и оптимизация структуры трансляционной таблицы с учетом фактического расположения строчных и прописных букв в кодовых таблицах символов. После завершения процедуры трансляции скан-кода в ASCII-код оба эти кода записываются в буфер клавиатуры.

Вспомогательные и виртуальные клавиши – расширенный скан-код.

Прикладные программы, предназначенные для обработки символьной информации, например, текстовые редакторы или специализированные программы верстки и дизайна книжной продукции, предлагают пользователям разнообразные «клавиатурные» приемы управления процессом редактирования, требующие использования не только буквенно-цифровых, но и вспомогательных клавиш.

Трудно представить себе текстовый редактор, не использующий клавиши навигации по тексту, клавиши *Tab*, *Enter*, *Back Space*, *Delete*, *Home* или *End*, но количество таких клавиш ограничено, и они далеко не покрывают всех потребностей профессионального редактирования текстов.

В этих условиях используют *виртуальные клавиши*, каждая из которых представлена комбинацией из нескольких клавиш, одновременно удерживаемых в нажатом состоянии. Чаще всего используют комбинации из двух клавиш, одна из которых – управляющая (*Shift*, *Ctrl* или *Alt*), а другая может быть как буквенно-цифровой, так и любой из вспомогательных, что позволяет в четыре раза увеличить количество «клавиш», физически не расширяя наборное поле клавиатуры. Каждая «виртуальная клавиша» может быть запрограммирована на выполнение определенной операции.

Информация о нажатых вспомогательных клавишах и комбинациях нажатых клавиш кодируется двухбайтовым двоичным числом – так называемым «расширенным скан-кодом», в котором младший байт содержит скан-код основной клавиши, а значение старшего байта определяется тем, с какой из управляющих клавиш она скомбинирована (в некоторых случаях – «0»).

Значения расширенных скан-кодов стандартизированы, процедура их формирования использует соответствующие трансляционные таблицы и учитывает состояние битовых флагов, связанных с клавишами *Shift*, *Alt* и *Ctrl*. Примеры расширенных скан-кодов приведены в таблице 7.3.

Расширенный скан-код хранится в буфере клавиатуры точно так же, как и пара «скан-код – ASCII-код» нажатой буквенно-цифровой клавиши. Такая система хранения предоставляет прикладным программам унифицированный доступ к символьным клавишам – как к основным буквенно-цифровым, так и к вспомогательным и виртуальным.

Таблица 7.3 – Примеры кодирования клавишей

Группа клавишей	Клавиша или комбинация клавишей	Port 60 _h	Буфер клавиатуры	
		Скан-код	ASCII	Расширенный скан-код
Управляющие клавиши	Left Shift	2A _h	-	-
	Right Shift	36 _h	-	-
	Left Ctrl	1D _h	-	-
	Right Ctrl	11D _h	-	-
	Alt	38 _h	-	-
	Caps Lock	3A _h		
	Num Lock	145 _h		
Буквенно-цифровые клавиши	1 / !	02 _h	31 _h / 21 _h	-
	2 / @	03 _h	32 _h / 40 _h	-
	3 / #	04 _h	33 _h / 23 _h	-
	q / Q	10 _h	71 _h / 51 _h	-
	w / W	11 _h	77 _h / 57 _h	-
	e / E	12 _h	65 _h / 45 _h	-
	r / R	13 _h	72 _h / 52 _h	-
	z / Z	2C _h	7A _h / 5A _h	-
Вспомогательные клавиши	END	4F _h	-	4FE0 _h
	HOME	47 _h	-	47E0 _h
	DEL	53 _h	-	53E0 _h
	Back Space	0E _h	-	0E08 _h
	Enter	1C _h	-	1C0D _h
	F11	57 _h	-	8500 _h
	F12	58 _h	-	8600 _h
Виртуальные клавиши (комбинации клавишей)	Alt+z	-	-	2C00 _h
	Ctrl+z	-	-	2C1A _h
	Ctrl+Enter	-	-	1C0A _h
	Ctrl+F11	-	-	8900 _h
	Ctrl+F12	-	-	8A00 _h
	Alt+F11	-	-	8B00 _h
	Flt +F12	-	-	8C00 _h
	Shift+F11	-	-	8700 _h
	Shift +F12	-	-	8800 _h

7.2.2.4 Алгоритм заполнения и чтения буфера клавиатуры

Буфер клавиатуры – это область ОЗУ, в которую обработчик 9-го прерывания записывает данные о нажатых символьных клавишах, и из которой прикладные программы считывают эти данные в порядке их записи. Для прикладной программы буфер клавиатуры – это устройство ввода, поставляющее программе (разумеется, по ее запросу) коды вводимых пользователем символов для их программной обработки.

Обмен данными между программами через буферную область памяти требует выполнения, как минимум, двух условий: во-первых, каждой из программ, участвующих в обмене, должен быть известен адрес расположения буферной области и, во-вторых, размер этой области должен быть достаточным для временного хранения блока передаваемых данных.

Обработчик 9-го прерывания и прикладная программа обмениваются блоками размером в 2 байта (скан-код + ASCII-код для основных буквенно-цифровых клавиш или расширенный скан-код для вспомогательных), поэтому двухбайтовой ячейки памяти было бы вполне достаточно для организации буфера обмена. Однако, в условиях, когда скорости процессов ручного ввода данных и их программной обработки могут существенно отличаться (как в ту, так и в другую стороны), использование такого «короткого» буфера неизбежно приводило бы к потерям времени на ожидание моментов освобождения или заполнения буфера.

В первых моделях ПК поддерживался буфер клавиатуры фиксированной длины в 32 байта, рассчитанный на регистрацию 16 последовательных нажатий клавиш и занимавший в области данных BIOS стандартный диапазон адресов ячеек памяти. В дальнейшем появилась возможность управления длиной и местом расположения буфера: его начальный и конечный адреса также указываются в соответствующих ячейках области данных BIOS (таблица 7.1).

Буфер клавиатуры, как структура данных – это циклический список типа «очередь», обслуживаемый по протоколу FIFO (First Input – First Output) – то есть чтение данных из буфера производится последовательно в порядке их записи, а по достижению конца очереди (как при чтении, так и при записи) осуществляется переход к ее началу.

Процессы записи и чтения буфера управляются двумя указателями: указатель «хвоста» очереди (*Tail*) определяет адрес очередной ячейки буфера, в которую обработчик прерывания должен записать блок данных о последней нажатой клавише, а указатель «головы» буфера (*Head*) – адрес ячейки буфера, из которой

прикладная программа должна прочитать первый из еще не прочитанных блоков данных. Оба указателя представляют сегментные адреса соответствующих ячеек буфера клавиатуры (хранится только часть адреса – смещение по сегменту 40h), для хранения каждого из них зарезервирована двухбайтовая ячейка в области данных BIOS (таблица 7.1).

До того, как будет введен первый символ, буфер клавиатуры находится в исходном состоянии, как показано на рисунке 7.5³⁰. В этом состоянии значения обоих указателей одинаковы и совпадают с начальным адресом буфера.

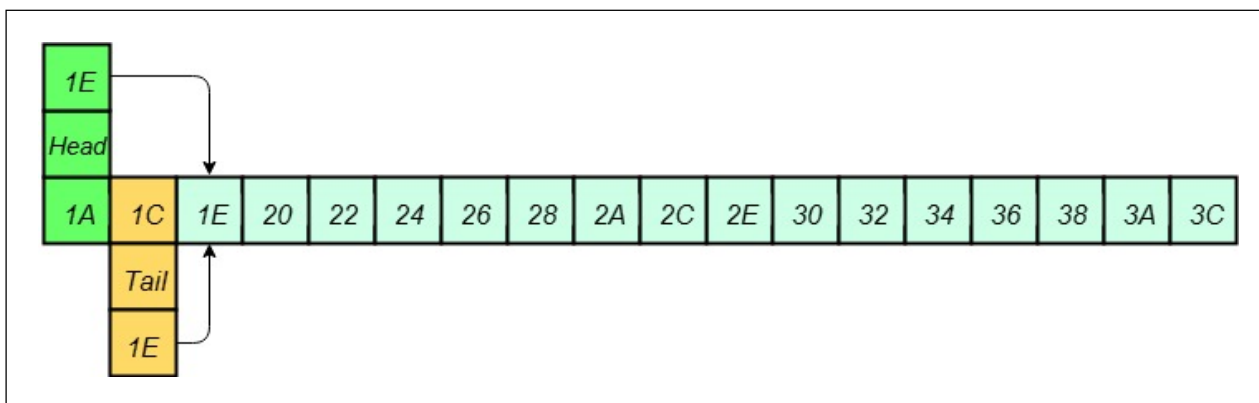


Рисунок 7.5 – Начальное состояние буфера клавиатуры

После завершения каждой операции записи данных в буфер программа-обработчик 9-го прерывания инкрементирует (увеличивает на два) значение «хвоста», а программа, читающая буфер, соответственно инкрементирует значение «головой» – таким образом, при вводе данных с клавиатуры хвост буфера постоянно «убегает вперед», а голова «догоняет».

Рисунок 7.6 иллюстрирует ситуацию, когда в буфер записаны данные о восьми нажатых символьных клавишах, а прочитаны пока только пять (очевидно, пользователь вводил данные быстрее, чем прикладная программа могла их обрабатывать).

³⁰ Все числа на рисунках 7.5 – 7.8 заданы в шестнадцатеричной системе счисления и представляют адреса (смещения по сегменту 40h) соответствующих ячеек буфера клавиатуры. Курсивом показаны адреса ячеек буфера, а жирным шрифтом – значения указателей головы и хвоста буфера, также являющиеся адресами.

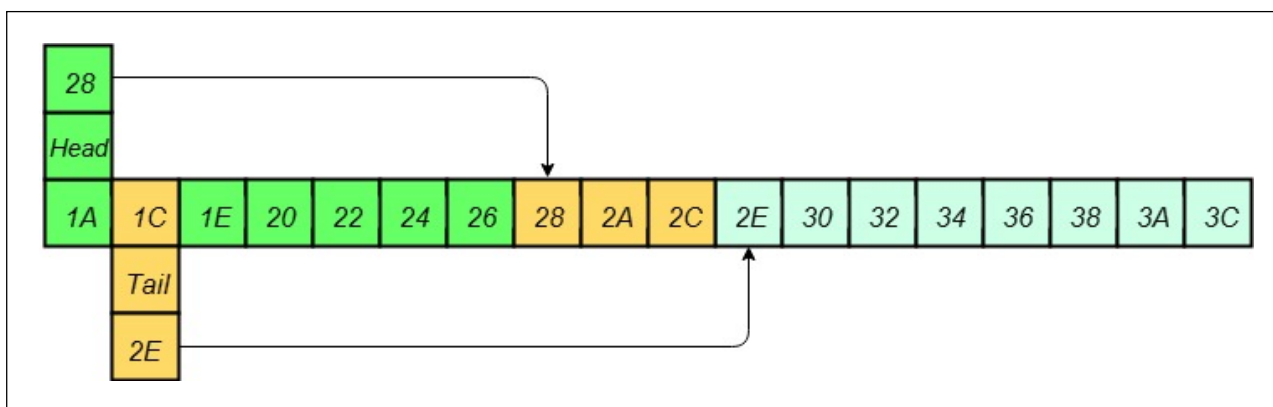


Рисунок 7.6 – Состояние «есть непрочитанные данные» буфера клавиатуры

Буфер частично заполнен, в нем есть место для записи новых блоков данных, также имеются данные, еще не прочитанные прикладной программой. Запись очередного блока данных в буфер будет производиться по адресу «хвоста» ($2E_h$), а чтение – по адресу «головы» (28_h).

Если в буфер ничего не записывается (пользователь устал писать и решил немного отдохнуть), то чтение буфера будет продолжено, и после каждого прочтения указатель головы будет инкрементирован на 2 – и так до тех пор, пока значение указателя головы не сравняется со значением хвоста буфера. В этот момент ($Tail = Head$) в буфере не останется непрочитанных данных (рисунок 7.7), будет идентифицирована ситуация «буфер логически пуст», и чтение должно быть прекращено.

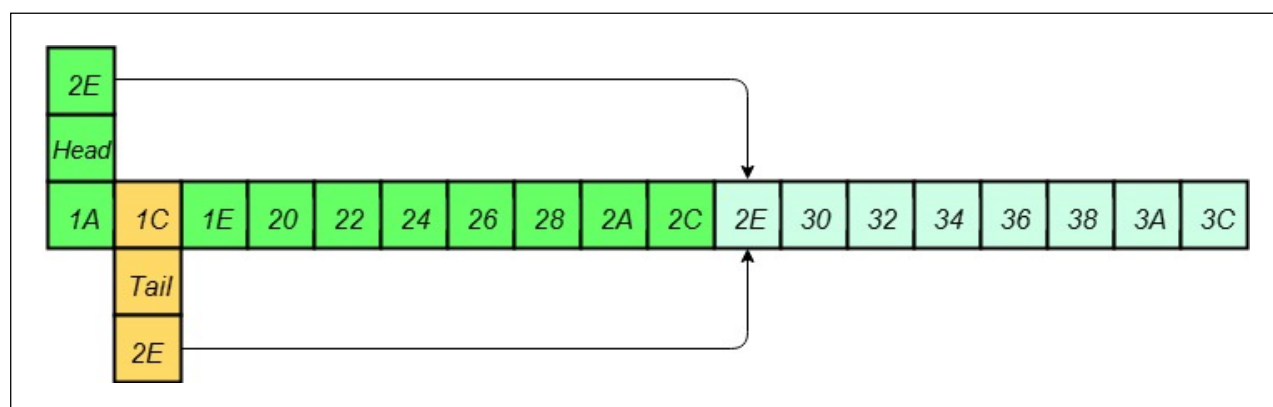


Рисунок 7.7 – Состояние «логически пуст» буфера клавиатуры

Буфер клавиатуры организован циклически: при достижении соответствующим указателем конца буфера (то есть, когда он станет равным $003E_h$ после очередного «+2») этот указатель устанавливается в начало буфера ($001E_h$).

Ситуация «буфер переполнен» иллюстрируется рисунком 7.8: указатель «головы» буфера остался в состоянии $2E_h$, так как теперь уже прикладная программа «взяла паузу» перед прочтением очередного блока данных (разумеется,

эта пауза нужна программе не для отдыха, а для обработки предыдущего блока введенных пользователем данных), а пользователь продолжает вводить данные и ввел очередные 16 символов.

Формальным идентификатором ситуации «буфер переполнен» является равенство « $| \text{Tail} - \text{Head} | = [\text{Длина буфера}]$ » с учетом циклической организации буфера. Попытка ввода данных об очередной нажатой клавише в переполненный буфер должна быть (и будет) заблокирована, о чем пользователь будет уведомлен звуковым сигналом.

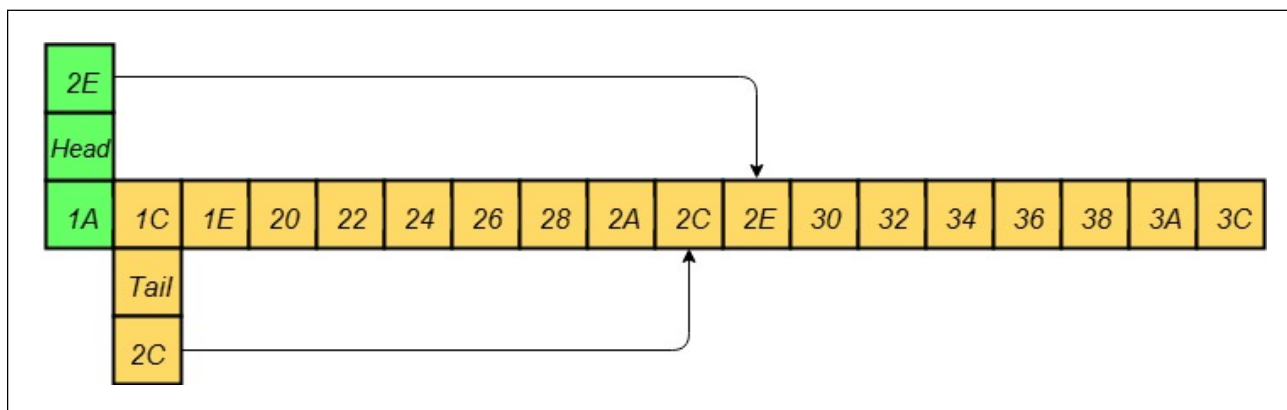


Рисунок 7.8 – Состояние «переполнения» буфера клавиатуры

Блок-схема алгоритма чтения данных из буфера соответствующей системной функцией (например, функцией №00_h программного прерывания INT16_h) приведена на рисунке 7.9, а блок-схема алгоритма реализации модуля «Запись кодов в буфер клавиатуры» (рисунок 7.4) программы обработки прерывания №9 – на рисунке 7.10.

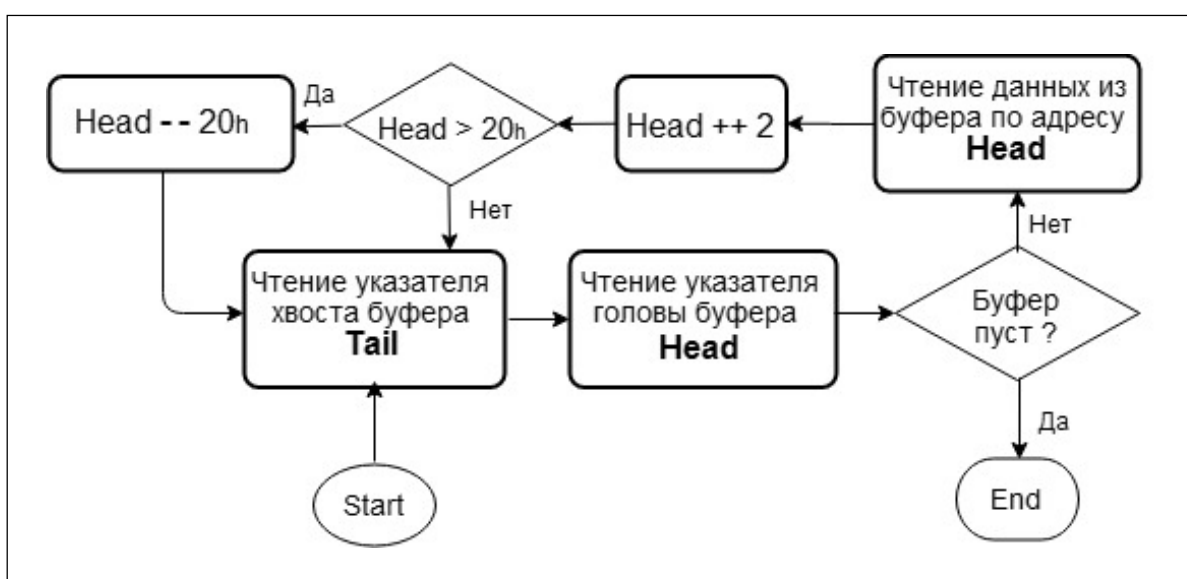


Рисунок 7.9 – Алгоритм чтения данных из буфера клавиатуры

Рассмотрим пример короткой программы (листинг 7.1), которая предлагает пользователю ввести с клавиатуры произвольный длинный текст и сохраняет результат ввода в переменной `kb_inp` строкового типа:

```
print('Enter a long text, as you want:')
kb_inp = input()
print(kb_inp)
```

Листинг 7.1 – Пример программы однократного ввода символьной строки

Функция `input()` циклически обращается к буферу клавиатуры (разумеется, используя для этого вызовы соответствующих системных функций). Пока пользователь не нажимает символьных клавиш, функция идентифицирует ситуацию «буфер логически пуст» ($Tail = Head$ на рисунке 7.6) и не считывает из буфера данные, расположенные по адресу *Head* (рисунок 7.8).

В момент нажатия очередной символьной клавиши центральный процессор прерывает работу прикладной программы и передает управление программе обработки прерывания №9, которая записывает в буфер код введенного символа, инкрементирует значение указателя хвоста буфера (рисунок 7.9) и заканчивает работу, возвращая управление прерванной прикладной программе.

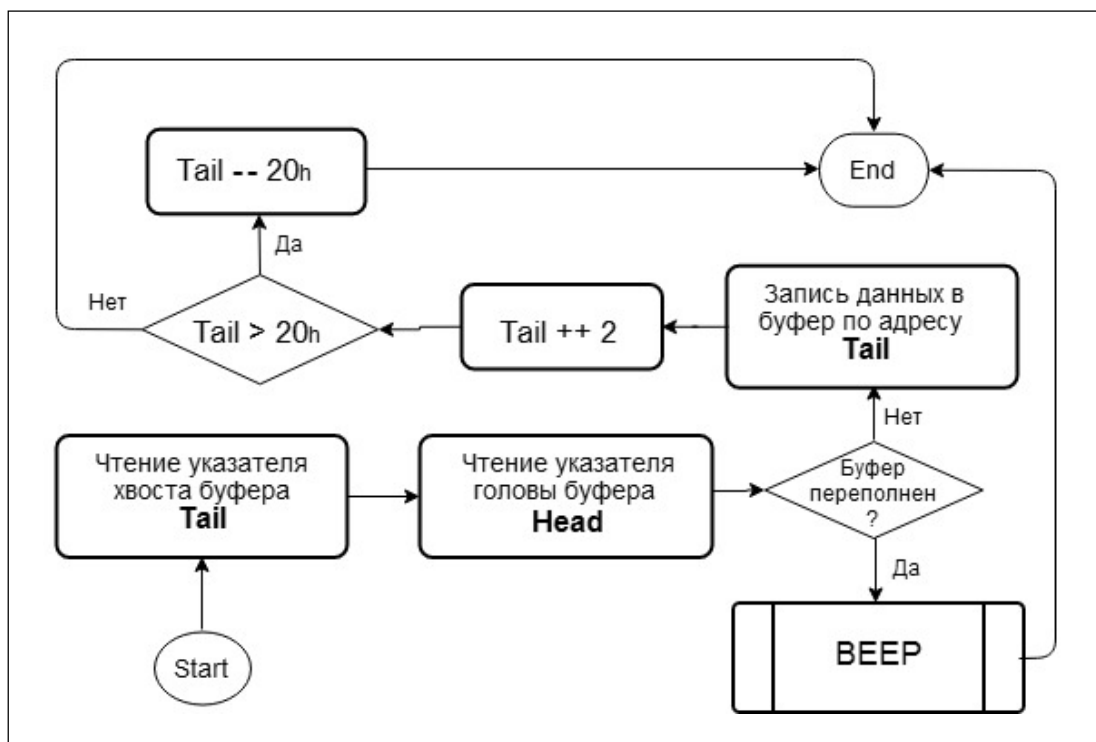


Рисунок 7.10 – Алгоритм записи данных в буфер клавиатуры

В новой ситуации буфер уже «не пуст», и прикладная программа читает код введенного символа по адресу головы буфера *Head*, сохраняет его в ячейке памяти, выделенной транслятором для объекта, связанного с переменной *kb_inp*, затем инкрементирует значение указателя головы буфера (*Head++2* на рисунке 7.8). В результате буфер клавиатуры снова оказывается в состоянии «логически пуст», и программа ожидает ввода очередного символа.

Процесс чтения буфера клавиатуры будет продолжаться до тех пор, пока пользователь не нажмет на клавишу *Enter*: прикладная программа, прочитав из буфера расширенный скан-код этой клавиши, завершит выполнение оператора `input()` и продолжит работу.

Заметим, что в такой ситуации пользователь может продолжать нажимать символьные клавиши – при каждом нажатии клавиши обработчик 9-го прерывания будет записывать в буфер клавиатуры очередной блок данных, но эти данные (введенные после нажатия клавиши *Enter*) уже не будут восприняты прикладной программой, приведенной на листинге 7.1.

Для подтверждения правильности последнего утверждения можно провести эксперимент с другой программой (листинг 7.2), в которой запрашивается ввод второй строки с 10-секундной задержкой после ввода первой.

```
import time
print('Enter a two long texts, as you want:')
first_kb_inp = input()
print(first_kb_inp)
time.sleep(10)
second_kb_inp = input()
print(second_kb_inp)
```

Листинг 7.2 – Пример программы ввода двух символьных строк

Если пользователь, не дожидаясь истечения 10 секунд, введет вторую строку, программа прочтет ее после истечения паузы – точно так же, как если бы эта строка была введена «вовремя».

7.2.2.5 Алгоритм прямого ввода кода символа

Любой символ 8-битовой кодовой таблицы, в том числе и не связанный ни с одной из клавиш клавиатуры, можно ввести, используя следующий прием: удерживая нажатой левую клавишу *Alt*, последовательно набрать ASCII-код символа в десятичной системе счисления, используя дополнительные цифровые клавиши (разумеется, при их наличии на клавиатуре ПК), и отпустить клавишу *Alt* после завершения ввода кода символа.

Для временного хранения вводимого ASCII-кода символа в области данных BIOS зарезервирована однобайтовая ячейка с фиксированным адресом $[0040:0019]_h$ (таблица 7.1). Обработчик 9-го прерывания, получив информацию о скан-коде нажатой дополнительной цифровой клавиши, анализирует состояние флагов клавиатуры и, если битовый флаг, соответствующий левой клавише *Alt*, установлен в единицу, определяет числовой двоичный код, соответствующий введенной десятичной цифре, затем суммирует его с числовым кодом, записанным в ячейке-накопителе ASCII-кода, записывает полученную сумму в эту же ячейку и заканчивает работу.

Под «суммированием» здесь понимается поразрядное суммирование со сдвигом: например, если в ячейке записан двоичный код числа 2, и нажата клавиша с цифрой 9, то после такого «суммирования» в ячейку будет записан двоичный код числа 29 ($2 \cdot 10 + 9$). При суммировании учитывается ограничение длины ячейки (1 байт), и все биты переполнения будут отброшены: например, если продолжить ввод и после цифры 9 ввести цифру 7, то после суммирования получим 297 ($29 \cdot 10 + 7$), что превышает максимально допустимое 255, и после отбрасывания битов переполнения в ячейке-накопителе кода останется двоичный эквивалент числа 42.

Таким образом, пока нажата левая клавиша *Alt*, в указанной ячейке будет «накапливаться» числовой код, который, после отпускания клавиши *Alt*, будет записан в очередную ячейку буфера клавиатуры в качестве ASCII-кода символа, при этом байт скан-кода в этой ячейке буфера получит нулевое значение.

7.3 Видеосистема ПК

Если клавиатура ПК – это основное устройство ручного ввода данных, то главным устройством вывода информации, представленной в визуальной форме, удобной для восприятия органами зрения человека, безусловно, является *дисплей*, часто называемый *видеомонитором*.

Экран дисплея является исполнительным устройством *видеосистемы* ПК, включающей комплекс программно-аппаратных средств и структур данных, обеспечивающих визуализацию информации, хранящейся в памяти компьютера в кодированной форме.

Основными техническими характеристиками видеосистемы ПК, определяющими качество экранного изображения, являются *разрешающая способность*, определяемая числом строк и столбцов точек (пикселей), выводимых на экран дисплея (для текстовых видеорежимов разрешающая способность оценивается также числом строк и столбцов символов на экране и размерами матрицы пикселей, используемой для изображения одного символа), и *цветность*, определяемая количеством цветов (или оттенков серого цвета) изображения, выводимого на экран. В текстовых режимах работы видеосистемы программируются цвет символа и цвет фона для каждого знакоместа, в графических – цвет каждого пикселя.

7.3.1 Аппаратный комплекс видеосистемы ПК

Аппаратный комплекс видеосистемы включает устройство отображения информации (дисплей) и видеоадаптер – программируемое устройство, управляющее дисплеем и обеспечивающее интерфейс с центральным процессором компьютера. С точки зрения программиста существенным является тип установленного видеоадаптера, режимы его работы, состав и назначение программируемых и информационных регистров.

Первые ПК серии IBM PC комплектовались монохромным дисплеем и видеоадаптером *MDA* (*Monochrome Display Adapter*), который работал только в текстовом режиме и имел разрешающую способность 640×350 пикселей. Позднее фирмой Hercules Computer Technology был разработан видеоадаптер *Hercules*, который, так же, как и *MDA*, являлся монохромным (двухцветным), однако имел более высокую разрешающую способность 720×350 пикселей и, что более существенно, обладал возможностью отображения графической информации.

История применения цветного экранного изображения в IBM-совместимых ПК начинается с видеоадаптера *CGA* (*Color Graphic Array*), который обеспечивал работу в текстовом и графическом режимах с 16-цветным изображением. При этом разрешающая способность *CGA* была весьма низкой (экран дисплея - 320×200, знакоместо символа – 8×8 пикселей) даже по сравнению с *MDA* и *Hercules*, что объясняется недостаточным для цветного изображения объемом видеопамяти (64 Кб), используемой этим видеоадаптером.

Позднее фирмой IBM были выпущены более совершенные видеоадаптеры *EGA* (*Enhanced Graphic Array*), *VGA* (*Video Graphic Array*) и *XGA* (*eXtended Graphic Array*) с более высокими техническими характеристиками (видеопамять – до 1 Мб, разрешение – до 1024×768, отображение до 65536 цветов).

К настоящему времени различными фирмами – производителями выпущено большое количество типов видеоадаптеров, существенно превосходящих *VGA* по своим техническим характеристикам, и получивших статус самостоятельных вычислителей, соизмеримых по мощности с центральными процессорами ПК. Эти видеоадаптеры принято объединять общим названием *SVGA* (*Super VGA*), и режимы их работы могут существенно отличаться от стандартных видеорежимов (таблица 7.4), разработанных в свое время фирмой IBM.

Таблица 7.4 – Характеристики стандартных режимов работы видеоадаптеров

№ режима	Тип	Разрешение	Число цветов	Видеоадаптеры	Начальный адрес
0, 0*, 0+	Текстовый	40×25 зн/м	16 (п/т)	CGA, EGA, VGA	[B800:0000] _h
1, 1*, 1+	Текстовый	40×25 зн/м	16		
2, 2*, 2+	Текстовый	80×25 зн/м	16 (п/т)		
3, 3*, 3+	Текстовый	80×25 зн/м	16		
4	Графический	320×200 пкс.	4		
5	Графический	320×200 пкс.	4(п/т)		
6	Графический	640×200 пкс.	2		
7	Текстовый	80×25 зн/м	2	MDA, EGA, VGA	[B000:0000] _h
8 – 0Ch	Резерв				
0Dh	Графический	320×200 пкс.	16	EGA, VGA	[A000:0000] _h
0Eh	Графический	640×200 пкс.	16		
0Fh	Графический	640×350 пкс.	4		
10h	Графический	640×350 пкс.	16		
11h	Графический	640×480 пкс.	2	VGA	
12h	Графический	640×480 пкс.	16		
13h	Графический	320×200	256		

Примечания:

1. Режимы 0, 2 и 5 являются режимами с подавлением цвета. В этих режимах вместо цветного выводится полутоновое изображение с заменой множества цветов на такое же количество оттенков серого цвета.
2. Разрешающая способность видеосистемы в текстовых режимах оценивается количеством столбцов и строк знакомест на экране и размерами матрицы пикселей, описывающей одно знакоместо: для CGA – 8×8, для EGA – 8×14, для VGA – 9×16.
3. Режимы 0*, 1*, 2* и 3* (EGA) - это аналоги текстовых режимов 0 ... 3 CGA, отличающиеся размерами матрицы описания символов (8×14).
4. Режимы 0+, 1+, 2+ и 3+ (VGA) - это аналоги текстовых режимов 0 ... 3 CGA, отличающиеся размерами матрицы описания символов (9×16).

7.3.2 Программное обеспечение видеосистемы ПК

Базовое программное обеспечение видеосистемы составляет набор функций прерывания BIOS INT10_h, обеспечивающих управление видеоадаптерами в стандартных режимах. Перечень некоторых функций с краткими комментариями по их применению приведен в таблице 7.5.

Таблица 7.5 – Видео-функции прерывания INT10_h

Функции	Назначение	Вход	Комментарии
00 _h	Выбор режима работы видеоадаптера	AH = 00 _h AL = № видеорежима	
01 _h	Изменение размеров курсора	AH = 01 _h CH = верхняя граница курсора CL = нижняя граница курсора	CH: биты 0-3 задают положение верхней границы курсора (0 – 15); биты 4-5 задают тип курсора – 00 – обычный, 01 – невидимый, 10 – мигающий, 11 – быстро мигающий.
02 _h	Изменение положения курсора	AH = 02 _h BH = № страницы DH = № строки DL = № столбца	При активизации страницы курсор устанавливается в заданную позицию.
09 _h	Запись символов	AH = 09 _h AL = ASCII-код символа BH = № страницы BL = байт атрибутов CX = количество символов	Записываются одинаковые символы с одинаковыми атрибутами в заданную страницу видеопамати. Запись производится с позиции, заданной текущим положением курсора. После завершения операции положение курсора не изменяется.

На листинге 7.3 приведен фрагмент ассемблерной программы, вызывающей функцию №00_h 10-го прерывания для выбора 4-го режима работы видеоадаптера (номер функции записывается в старший байт AH регистра A процессора, номер режима записывается в младший байт AL этого же регистра и затем вызывается программа обработки 10-го прерывания):

```
mov AH,0; mov AL,4; int 10h.
```

Листинг 7.3 – пример вызова видео-функции

7.3.3 Структуры данных, используемые видеосистемой ПК

Основной структурой данных, обслуживающей процесс вывода информации на экран дисплея, является так называемая *видеопамять*, выполняющая роль буфера обмена между программами, выполняемыми центральным процессором, и видеоадаптером, управляющим дисплеем. Функционирование видеосистемы обеспечивается также дополнительными структурами данных (таблица 7.6), организованными в ОЗУ и ПЗУ.

Видеопамять - основная структура данных, выполняющая функции буфера обмена между прикладными программами и видеоадаптером. Видеопамять логически расположена в основном адресном пространстве в диапазоне адресов с [A000:0000]_h по [B000:FFFF]_h (128 Кбайт). Физически видеопамять расположена на плате видеоадаптера и может иметь объем, многократно превышающий 128 Кбайт. Видеопамять логически разделена на множество так называемых видеостраниц – последовательно расположенных областей, объем каждой из которых достаточен для представления информации, отображаемой на одном полном экране. Структура видеопамяти, ее объем и расположение определяются типом видеоадаптера и режимом его работы.

Переменные BIOS. Для обслуживания видеосистемы используются часть *области данных BIOS*, формируемой в процессе инициализации (загрузки DOS). Знание адресов этих переменных позволяет определять количество и типы установленных видеоадаптеров, режимы их работы, объем видеопамяти и ряд других параметров видеосистемы.

Таблица 7.6 – Структуры данных, обслуживающих видеосистему ПК

Начальный адрес	Длина, байт	Назначение области памяти
Таблица векторов прерываний		
[0000:0040] _h	4	INT 10 _h – указатель на видео-функции BIOS
[0000:007C] _h	4	INT 1F _h – указатель на таблицу знакогенератора для символов с кодами 128 – 255
[0000:010C] _h	4	INT 43 _h – указатель на таблицы знакогенератора для символов с кодами 0 – 255 (EGA/VGA)
Область данных BIOS		
[0040:0010] _h	1	Флаги конфигурации. 5-й и 4-й биты определяют тип видеоадаптера: 00 – EGA; 01 – CGA 40 × 25; 10 – CGA 80 × 25; 11 – MDA
[0040:0049] _h	1	Номер текущего видеорежима
[0040:004A] _h	2	Число символов в строке
[0040:004C] _h	2	Размер видеостраницы (в байтах)
[0040:004E] _h	2	Начальный адрес активной видеостраницы (смещение в видео-сегменте)
[0040:0050] _h	8 × 2	Координаты курсора в каждой из 8 страниц: Младший байт - № колонки, старший - № строки
[0040:0060] _h	2	Размер (форма) курсора: младший байт – № последней линии, старший байт – № первой линии
[0040:0062] _h	1	Номер активной страницы видеопамати
[0040:0065] _h	1	Данные регистра режима CGA
[0040:0066] _h	1	Данные регистра цветовой палитры CGA
[0040:0084] _h	1	Число текстовых строк экрана минус единица
[0040:0085] _h	2	Высота символа в пикселях
[0040:0087] _h	1	1-й байт данных о EGA. 6-й и 5-й биты определяют объем установленной видеопамати: 00 - 64К, 01 - 128К, 10 – 192К, 11 – 256К
[0040:0088] _h	1	2-й байт данных о EGA
[0040:00A8] _h	2	Адрес таблицы окружения минус единица
Видеопамать		
[A000:0000] _h	64Кб	Видеопамать в графических режимах EGA, VGA
[B000:0000] _h	32Кб	Видеопамать в монохромном текстовом режиме MDA
[B800:0000] _h	32Кб	Видеопамать в цветковых текстовых режимах и в графическом режиме CGA
ПЗУ		
[C000:0000] _h	16Кб	ROM BIOS EGA / VGA

Таблица окружения. Содержит восемь адресов – указателей на различные таблицы и буфера данных, используемые функциями BIOS, обслуживающих видеосистему: таблицу параметров, область сохранения, вспомогательные таблицы символов для текстовых и графических видеорежимов. Если элемент таблицы окружения равен нулю, то соответствующий блок данных не используется. Указатель на таблицу окружения хранится в области данных BIOS по адресу [0040:00A8]_h.

Таблицы знакогенераторов – специальные структуры данных, располагающиеся в ОЗУ и ПЗУ и используемые видеоадаптерами в процессе формирования «точечных» образов символов на экране. Указатели на таблицы знакогенераторов - *векторы прерываний* INT 1F_h и INT 43_h. Различные программы (например, программы-русификаторы) могут загружать собственные знакогенераторы и переустанавливать на них соответствующие векторы прерываний.

7.3.4 Кодирование данных в видеопамяти

В соответствии с установленным режимом работы видеосистемы в видеопамяти ПК организуется множество *страниц*, каждая из которых может использоваться для хранения образа полного экрана дисплея. Одна из страниц объявляется *активной*, и ее номер записывается в области данных BIOS по адресу [0040:0062]_h.

Прикладная программа формирует образ экрана и, используя справочную информацию области данных BIOS (таблица 7.6), записывает кодированные данные в активную страницу видеопамяти. Видеоадаптер циклически считывает данные активной видеостраницы, преобразует (декодирует) их и формирует соответствующие сигналы управления дисплеем.

Размер видеостраницы определяется типом видеорежима, разрешающей способностью и числом отображаемых цветов, а количество страниц ограничивается объемом установленной видеопамяти. Все видеостраницы пронумерованы, начиная с нулевой.

В текстовых и графических режимах используются различные системы кодирования элементов экрана, так как в первом случае в качестве элемента экрана используется *знакоместо*, а во втором – *пиксель*.

7.3.4.1 Кодирование данных в текстовых режимах

Знакоместо – это прямоугольная область экрана (8×8, 8×14 или 8×16 пикселей в зависимости от типа видеоадаптера), внутри которой точно отображается символ, ASCII-код которого записан в видеопамять по соответствующему адресу.

Каждое знакоместо экрана кодируется двухбайтовым машинным словом, в котором младший байт – это *байт символа*, а старший – *байт атрибута*. Байт символа содержит ASCII-код символа, выводимого на экран, а байт атрибута задает цвет символа и цвет фона. Однобайтовая система кодирования цвета позволяет описать 256 (2^8) различных цветовых образов кодируемого знакоместа: младшие 4 бита кодируют цвет символа, а старшие 4 бита – цвет фона.

Система кодирования цвета знакоместа (рисунок 7.11) поддерживает модель **RGB** (**R**ed – **G**reen – **B**lue), в которой для кодирования цвета отводится по 4 бита: 3 младших бита – цветовые, а старший бит – это бит интенсивности выбранного цвета. Такая модель позволяет закодировать 16 различных цветов символа и столько же цветов фона: 8 различных комбинаций красного, зеленого и синего цветов, смешанных в равных пропорциях пониженной яркости (при нулевом значении бита интенсивности), и столько же соответствующих ярких цветов (при единичном значении бита интенсивности). 7-й бит байта атрибута может «отвечать» либо за интенсивность фона (*i*), либо за его мигание (*b*), в зависимости от установки параметров видеоадаптера.

Цвет фона				Цвет символа			
7	6	5	4	3	2	1	0
<i>i/b</i>	R	G	B	<i>i</i>	R	G	B

Рисунок 7.11 – Формат байта атрибута символа

Такая система кодирования позволяет «пронумеровать» цвета символа и цвета фона целыми числами в диапазоне от 0 до 15, что используется во многих языках программирования при выводе символов на экран.

Пусть, например, переменные *Color* и *Background* (типа «целое без знака») получили значения $Color = 12$ и $Background = 9$, тогда переменная *Attrib* (того же типа), представляющая атрибут символа, записываемый в видеопамять, получит значение $Attrib = Color + 16 \times Background = 156$. В результате в соответствующем

знакоместе экрана будет отображен символ ярко-красного цвета ($Color = 12 = 1100_2$) на синем мигающем фоне ($Background = 9 = 1001_2$).

Размер видеостраницы определяется общим количеством знакомест на экране: например, для стандартных видеорежимов №2 и №3 (таблица 7.4) требуется видеостраница размером 4000 байтов (25 строк и 80 столбцов знакомест, по 2 байта на знакоместо). Фактически под видеостраницу отводится 4 Кб (4096 байтов, или 1000_h в шестнадцатеричной системе счисления), из которых последние 96 байтов не используются.

Для текстовых видеорежимов (таблица 7.4) выделено всего 32 Кб видеопамати (в сегменте $B800_h$ для CGA-совместимых видеорежимов и в сегменте $B000_h$ – для MDA), что позволяет организовать до 8 видеостраниц (для стандартных видеорежимов №2, №3 и №7, и до 16 страниц – для режимов №0 и №1).

Так, для видеорежимов №2 и №3 нулевая страница будет занимать диапазон адресов с $[B800:0000]_h$ по $[B800:0FFF]_h$, и каждая последующая страница будет смещена на 4096 байтов относительно предыдущей: 1-я страница – с $[B800:1000]_h$ по $[B800:1FFF]_h$, 2-я – с $[B800:2000]_h$ по $[B800:2FFF]_h$ и т.д.

Зная размер видеостраниц в байтах L и номер видеостраницы N , можно определить ее начальный адрес (смещение от начала соответствующего сегмента видеопамати): $[N \times L]_h$.

При двухбайтовой системе кодирования адрес (смещение от начала видеостраницы) машинного слова, представляющего в видеопамати некоторое знакоместо, вычисляется по формуле $2 \times (y \times m + x)$, где x и y – координаты знакомого места на экране (номер столбца и строки, которые отсчитываются слева направо и сверху вниз и нумеруются с нуля), а m – количество символов в строке, определяемое выбранным видеорежимом (таблица 7.4).

Все эти данные (N , L , m , координаты курсора x и y , а также номер видеорежима, определяющий номер сегмента памяти, выделенный видеоадаптеру) хранятся в соответствующих *переменных BIOS* (таблица 7.6).

Процесс вывода символа на экран реализуется по следующей схеме:

- 1) Программа (например, текстовый редактор или интерпретатор командной строки) обращается к области данных BIOS (таблица 7.6) и определяет номер активной страницы N , ее размер в байтах L , координаты курсора x и y и число символов в строке m .
- 2) По этим данным вычисляется адрес машинного слова, описывающего в видеопамати знакоместо, на которое указывает курсор. Например,

для видеорежима №2 и нулевой видеостраницы левое верхнее знакоместо экрана с координатами [0,0] отображается на ячейку видеопамати с адресом [B800:0000]_h, а правое нижнее знакоместо с координатами [79,24] – [B800: 0F9E]_h.

- 3) По вычисленному адресу программа записывает в видеопамать байт символа и байт атрибутов (байт символа, содержащий его ASCII-код, до этого мог быть прочитан программой в буфере клавиатуры, а байт атрибута, описывающий цвета символа и фона, сгенерирован программой).

Байт символа (ASCII-код) используется далее видеоадаптером для вычисления адреса расположения соответствующей битовой матрицы в таблице знакогенератора (п. 7.3.5), а байт атрибутов – для формирования сигнала управления цветом изображения символа.

7.3.4.2 Кодирование данных в графических режимах

В графических режимах (таблица 7.4) минимальной «единицей» экранного пространства является один пиксель (в отличии от знакоместа, описываемого матрицей пикселей в текстовых режимах), и единственным кодируемым параметром пикселя является его цвет. Размер блока видеопамати, описывающего один пиксель, определяется количеством отображаемых цветов (или градаций серого цвета). Например, в монохромных (двухцветных) режимах №6 и №11_h для представления пикселя достаточно одного бита, в 4-цветных режимах №4, №5 и №0F_h – двух битов, в 16-цветных режимах №0D, №0E, №10_h и №12_h – четырех битов, а в 256-цветном режиме №13_h для описания одного пикселя потребуется один байт.

Таким образом, одним байтом видеопамати может быть описано от 1 до 8 пикселей, причем старшие биты байта отвечают за «левые» пиксели, а младшие – за «правые». Например, цвет левого верхнего пикселя экрана (с координатами [0,0]) кодируется следующими битами нулевого байта видеостраницы:

- в режимах №6 и 11_h - старшим битом нулевого байта;
- в режимах №4 и №5 - двумя старшими битами нулевого байта;
- а в режиме №13_h – всеми восемью битами нулевого байта.

Двоичное число, описывающее пиксель в графическом режиме – это числовой код, определяющий «номер» цвета пикселя в соответствии с загруженной в видеоадаптер цветовой палитрой.

Размер видеостраницы в графических видеорежимах определяется разрешающей способностью (количеством пикселей на экране) и размером блока, описывающего один пиксель. Например, в режимах №4 и №5 (4 цвета, разрешение 320×200) и в режиме №6 (2 цвета, разрешение 640×200) для одной видеостраницы потребуется 16000 байт (фактически – 16 Кб). В этих режимах объема видеопамати достаточно для размещения двух страниц, однако адаптеры CGA, EGA и VGA поддерживают одну страницу с начальным адресом B800:0000. В режиме №13_h (256 цветов, разрешение 320×200) одна страница занимает 64 Кбайт, то есть целый банк видеопамати, располагающийся с адреса A000:0000.

7.3.5 Знакогенераторы

Знакогенератор (или таблица знакогенератора) – это специальная структура данных, описывающая растровое (точечное) изображение символа и используемая видеоадаптером для преобразования ASCII-кода символа в последовательность сигналов, управляющих состоянием пикселей. Графический образ каждого символа описывается битовой матрицей, размерность которой определяется разрешающей способностью видеосистемы и различна для разных видеоадаптеров (CGA – 8×8, EGA – 8×14, VGA – 8×16). Каждый узел матрицы описывается одним битом, при этом единичное значение бита соответствует активному (светлому) пикселю, а нулевое – пассивному (темному).

Иллюстрация системы числового двоичного кодирования графических образов символа для видеоадаптера CGA приведена на рисунке 7.12. Как видно из рисунка, для кодирования образа одного символа матрицей 8×8 требуется блок памяти размером в 8 байтов – по одному байту на каждую строку матрицы. При использовании матрицы 8×14 (EGA) потребуется 14 последовательных байтов, а при использовании матрицы 8×16 (VGA) – 16 байтов (по одному байту на каждую строку матрицы). Количество строк пикселей k , используемых для представления символа в знакоместе, хранится в соответствующей ячейке области данных BIOS ([0040:0085]_h, таблица 7.6).

В таблице знакогенератора блоки описаний символов расположены линейно в порядке возрастания их ASCII-кодов, что позволяет вычислять начальные адреса блоков (смещение относительно начала таблицы знакогенератора) по простой формуле: $k \times \text{ASCII-код}$, где k – количество байтов, отводимых в таблице знакогенератора для описания одного символа.

		Образ символа								Коды строк матрицы	
0										0000 0000	00 _h
1					■	■	■			0000 1110	0E _h
2				■				■		0001 0010	12 _h
3			■					■		0010 0010	22 _h
4			■					■		0011 1110	3E _h
5			■	■	■	■	■	■		0010 0010	22 _h
6			■					■		0010 0010	22 _h
7			■					■		0010 0010	22 _h
		7	6	5	4	3	2	1	0		

Рисунок 7.12 – Представление символа «А» битовой матрицей 8×8

Начальный адрес таблицы знакогенератора EGA / VGA для символов с кодами от 0 до 255 хранятся в ROM BIOS и доступен по вектору прерывания 43_h. EGA поддерживает 4 различных таблицы, а VGA – 8 таблиц, при этом одновременно могут быть активны две любые таблицы, что позволяет отображать на экране до 512 различных символов. Номера активных таблиц определяются содержимым регистра выбора знакогенератора видеоадаптера.

Начальный адрес таблицы знакогенератора для 128 символов «национальных алфавитов» с ASCII-кодами от 128 до 255 доступен по вектору прерывания 1F_h, что используется программами-«русификаторами», которые формируют битовые матрицы для каждого символа соответствующего алфавита, записывают их в определенную область ОЗУ и соответственно переустанавливают вектор прерывания 1F_h.

BIOS EGA/VGA содержит специальную функцию, управляющую загрузкой шрифтов (функция 11_h прерывания INT 10_h).

7.4 Контрольные задания

Система обработки прерываний

Задание 7.1: Определите термины: *прерывание, аппаратное прерывание, программное прерывание.*

Задание 7.2: Какие основные функции выполняет контроллер прерываний?

Задание 7.3: Определите термины: *IRQ-линия прерывания, номер прерывания, вектор прерывания.*

Задание 7.4: Используя электронный справочник *HELP*, определите назначение и использование следующих прерываний: INT 00_h, INT 08_h, INT 09_h, INT 10_h, INT 16_h, INT 1F_h, INT 21_h, INT 43_h:

- какие из перечисленных прерываний являются программными, какие – аппаратными?
- определите DOS-функции 21-го прерывания, реализующие файловые операции, рассмотренные в п. 6.2 учебного пособия.

Задание 7.5: Определите *адреса расположения векторов прерываний*, номера которых указаны в задании №7.4.

Задание 7.1: Используя фрагмент таблицы векторов прерываний, приведенный на рисунке 7.1, определите *значения векторов прерываний*, номера которых указаны в задании №7.4.

Задание 7.2: На какие области памяти (таблица 4.1) указывают векторы прерываний №1F_h и №43_h?

Задание 7.3: Повторите задания №7.6 и №7.7, используя одну из программ-анализаторов памяти, установленную на Вашем рабочем компьютере. Сравните и объясните полученные результаты.

Задание 7.4: * Разработайте программную модель таблицы векторов прерываний, реализующую метод чтения вектора прерывания по его номеру.

Задание 7.5: * Разработайте программную модель стека, реализующую методы записи данных в стек и чтения данных из стека.

Клавиатура ПК

Задание 7.6: Перечислите основные функции контроллера клавиатуры. С какими компонентами ПК он взаимодействует и с какими целями?

Задание 7.7: Какая информация закодирована scan-кодом клавиши?

Задание 7.8: Какие манипуляции с клавишами не изменяют состояния «флагов клавиатуры», а какие – состояния «буфера клавиатуры»?

Задание 7.9: Как программно идентифицировать ситуации «буфер клавиатуры переполнен» и «буфер клавиатуры пуст»?

Задание 7.10: Всегда ли пуст «пустой» буфер клавиатуры?

Задание 7.11: В каком формате хранятся указатели «головой» и «хвоста» буфера клавиатуры? Какие программы должны считывать и/или модифицировать значения этих указателей и с какими целями?

Задание 7.12: * Разработайте упрощенную программную модель буфера клавиатуры (хранение только ASCII-кодов) и реализуйте на этой модели следующие методы:

- 1) `buf_tail()` для определения указателя «хвоста» буфера;
- 2) `buf_head()` для определения указателя «головой» буфера;
- 3) `buf_full()` для определения состояния «буфер переполнен»;
- 4) `buf_empty()` для определения состояния «буфер пуст»;
- 5) `buf_write(code)` запись в буфер кода очередного символа;
- 6) `buf_read()` чтение из буфера кода последнего введенного символа.

Видеосистема ПК

Задание 7.13: Определите цвет символа и цвет фона для следующих (десятичных) значений байта атрибута (стандартный видеорежим №3): 7, 128, 156, 255.

Задание 7.14: Определите значение байта атрибута (стандартный текстовый видеорежим №3) для отображения символа ярко-красным цветом на зеленом мигающем фоне и бледно-зеленым цветом на красном фоне.

Задание 7.15: Определите понятие «видеостраница». Какой объем видеостраницы необходим для реализации видеорежима №3?

Задание 7.16: Определите диапазон адресов ячеек видеопамати для видеостраницы №4 (стандартный видеорежим №3).

Задание 7.17: Определите адреса ячеек 2-й страницы видеопамати, выделенных для хранения символа в знакоместе экрана с координатами $[20_h; 10_h]$ (стандартный видеорежим №3).

Задание 7.18: Как программно изменить форму курсора и определить текущее положение курсора на видеостранице с заданным номером?

Задание 7.19: Для каких целей зарезервированы прерывания INT_1F и INT_43h?

Задание 7.20: Определите объем памяти, необходимый для хранения таблицы знакогенератора (256 символов в формате 9×16).

Задание 7.21: * Создайте программную модель байта атрибута (стандартный видеорежим №3) и реализуйте на базе этой модели следующие методы:

- 1) `color(attrib)` определение цвета символа по его атрибуту;
- 2) `background(attrib)` определение цвета фона по его атрибуту;
- 3) `attrib(color, background)` для определения значения атрибута по заданным цветам символа и фона.

Задание 7.22: * Создайте упрощенную программную модель таблицы знакогенератора (16 любых символов Вашего персонального алфавита в формате 8×8 пикселей) и реализуйте на базе этой модели следующие методы:

- 1) `write_sign(code, matrix)` для заполнения фрагмента знакогенератора, соответствующего символу с кодом `code`, данными битовой матрицы, представленными параметром `matrix`;
- 2) `print_sign(code)` для отображения графического образа символа по его коду (подобно тому, как это показано на рисунке 7.12).

Задание 7.23: ** Напишите программу для отображения графического образа символа по его коду (рисунок 7.12), использующую реальную таблицу знакогенератора, адресуемую вектором прерывания INT 43h.

Общие методические указания

Структура и содержание

Лабораторный практикум включает шесть лабораторных работ по тематике 6-й и 7-й глав учебного пособия.

Лабораторные работы №1, №2 и №3 нацелены на изучение файловой FAT-системы ПК, основные концепции которой изложены в разделах 6.1, 6.2.1 и 6.2.2 учебного пособия. При выполнении этих работ практически осваивается язык командной строки и исследуются алгоритмы реализации типовых файловых операций.

Основная задача 4-й лабораторной работы – освоение программных анализаторов адресного пространства в процессе исследования структуры двух системных областей ОЗУ: таблицы векторов прерываний и области данных BIOS. При подготовке к выполнению этой работы потребуется изучение материала, изложенного в разделах 4.6 и 7.1 учебного пособия.

В лабораторной работе №5 исследуется процесс обмена данными с клавиатурой ПК, детально рассмотренный в разделе 7.2 учебного пособия. Анализируются системные структуры данных и алгоритмы доступа к буферу клавиатуры, реализуемые программной обработкой прерывания №9.

Завершает лабораторный практикум работа №6, в которой практически исследуются система кодирования видео-данных в текстовых режимах работы видеосистемы ПК и структура таблиц знакогенераторов. При подготовке к выполнению этой работы потребуется изучение материала, изложенного в разделах 7.3.3, 7.3.4 и 7.3.5 учебного пособия.

Каждая лабораторная работа содержит несколько взаимосвязанных практических заданий экспериментального характера, выполнение которых направлено на решение поставленных в работе задач и требует освоения и применения соответствующих инструментальных программных средств.

Программное обеспечение.

Все лабораторные работы выполняются на виртуальной DOS-машине со следующими характеристиками: 640 Mb RAM; 128 Mb VideoRAM; 256 Mb ROM; 1,44 Mb FDD (FAT-12); 600 Mb HDD (FAT-16); MS DOS 5.5.

В среде виртуальной DOS-машины доступно следующее инструментальное программное обеспечение:

1. *Norton Commander* и *DOS Navigator* – программные оболочки со встроенными текстовыми редакторами, средствами управления файлами и DOS-командами.
2. *HELP* – электронный справочник по MS DOS.
3. *DiskEdit* – программный анализатор/редактор дискового пространства.
4. *Peek* и *FxShow* – резидентные программные анализаторы/редакторы адресного пространства ОЗУ и ПЗУ.
5. *Calc* – резидентный калькулятор с поддержкой двоичной, десятичной, и шестнадцатеричной систем счисления.

Инструкция по установке виртуальной DOS-машины на рабочий ПК приведена в приложении А.

Требования к отчетной документации

По каждой выполненной работе оформляется отдельный отчет (представляется к защите в виде файла текстового формата).

Отчет по лабораторной работе должен содержать:

- титульный лист (кафедра, дисциплина, номер и наименование работы, исполнитель, преподаватель, дата представления отчета);
- цели и задачи, описание методики проведения работы, перечень используемых инструментальных средств;
- результаты выполнения индивидуальных практических заданий:
 - иллюстративные материалы (снимки экрана и/или иные графические материалы), поясняющие процесс и результаты выполнения работы;
 - анализ полученных результатов с собственными выводами исполнителя;
 - ответы на контрольные вопросы (при их наличии).

Защита результатов выполнения лабораторных работ

Все лабораторные работы выполняются индивидуально, защита работы проводится в форме собеседования по материалу представленного отчета.

В процессе защиты оценивается полнота и качество выполнения практических заданий, грамотность использования инструментальных средств, правильность и обоснованность выводов по результатам работы, качество оформления отчета.

Лабораторная работа №1.

Командный интерфейс пользователя ПК

Задачи выполнения лабораторной работы – изучение языка командной строки и приобретение практических навыков использования команд для выполнения типовых файловых операций.

Команда - это средство текстового общения пользователя с операционной системой компьютера. Команда вводится с клавиатуры и отображается в *командной строке* экрана. В процессе записи команды ее можно редактировать – до тех пор, пока не нажата клавиша *Enter*, после чего команда записывается в специальный буфер ОЗУ³¹, и начинается процесс ее обработки интерпретатором командной строки. В MS DOS функции интерпретации команд выполняла системная программа Command.com, в ОС Windows – приложение Cmd.exe.

Процесс интерпретации команд кратко описан в разделе 5.2.1 учебного пособия. Если команда введена корректно, будет выполнена соответствующая команде системная функция или прикладная программа, в противном случае будет выдано диагностическое сообщение.

1.1 Классификация команд

По способу исполнения различают *внутренние* и *внешние* команды, по функциональному назначению – команды управления томами, каталогами и файлами, а также служебные команды. Полный перечень команд с их краткими описаниями можно получить, выполнив внутреннюю команду *Help*, примеры некоторых команд различных категорий приведены в таблице 8.1.

Внутренние и внешние команды.

Внутренние команды исполняются соответствующими системными функциями, доступ к которым получает интерпретатор командной строки по имени команды, записываемой первой в командной строке. Количество внутренних команд

³¹ В буфере командной строки хранятся несколько введенных ранее команд в порядке их исполнения. Для извлечения команд из буфера в командную строку используются клавиши-стрелки: клавиши ↑ и ↓ для выбора очередной команды из списка исполненных команд и клавиша → для посимвольного выбора предыдущей команды. Если команды исполняются при активной программной оболочке Norton Commander (или любом из ее аналогов), извлечение очередной команды из буфера осуществляется комбинацией клавиш «Ctrl+E».

ограничено, их имена – это зарезервированные слова, обозначающие операции, для некоторых внутренних команд допускается использование их сокращенных имен, например: *DIR* (*Directory*), *DEL* (*Delete*), *REN* (*Rename*).

Таблица 8.1 Примеры команд различных типов и категорий

Категории команд	Имя команды	Тип команды	Выполняемая функция
Дисковые операции	имя_тома:	Внутренняя	Активизация тома (имя_тома – одна из букв от A до Z).
	LABEL	Внешняя	Отображение и/или редактирование метки тома
	VOL	Внутренняя	Отображение метки тома
	CHKDSK	Внешняя	Проверка состояния структуры диска (файлы, каталоги, FAT)
	FDISK	Внешняя	Разбиение жесткого диска на логические разделы (тома)
	FORMAT	Внешняя	Форматирование тома
Операции с каталогами	DIR	Внутренняя	Вывод оглавления каталога
	CD	Внутренняя	Изменение текущего каталога
	MD	Внутренняя	Создание нового каталога
	RD	Внутренняя	Удаление каталога
	TREE	Внешняя	Вывод "дерева каталогов"
Операции с файлами	COPY	Внутренняя	Копирование файлов
	RENAME	Внутренняя	Переименование файлов
	TYPE	Внутренняя	Просмотр текстового файла
	MORE	Внешняя	Постраничный просмотр файлов
	EDIT	Внешняя	Редактирование текстовых файлов
	DEL	Внутренняя	Удаление файлов
	ERASE	Внутренняя	Удаление файлов
	PRINT	Внешняя	Печать файлов
	XCOPY	Внешняя	Копирование групп файлов вместе со структурой (деревом) каталогов
REPLACE	Внешняя	Замена файлов каталога одноименными файлами из другого каталога	
Служебные команды	DATE	Внутренняя	Установка системной даты
	TIME	Внутренняя	Установка системного времени
	PROMPT	Внутренняя	Установка формы «приглашения DOS»
	PATH	Внутренняя	Установка пути поиска программных файлов
	VER	Внутренняя	Вывод версии ОС

В качестве *внешней команды* используется *имя исполнимого файла* (.com, .exe или .bat). При выполнении внешней команды производится поиск указанного в команде файла, загрузка его в память компьютера и запуск на выполнение.

Имя команды включает *спецификацию* исполнимого файла в соответствии с принятым стандартом: **имя_тома:\путь\имя_файла.расширение**. При этом обязательным является только параметр **имя_файла**:

- **имя_тома** может быть опущено, если исполнимый файл находится на активном томе;
- **путь** к файлу³² может не указываться в случае, если исполнимый файл зарегистрирован в текущем каталоге или, если путь к родительскому каталогу этого файла был предварительно задан командой **PATH**;
- **расширение** имени файла (.com, .exe или .bat) может не указываться в случае, если в целевом каталоге отсутствуют одноименные исполнимые файлы.

1.2 Формат команд

Команда – это текстовая строка, содержащая три компонента:

< имя команды > < параметры команды > / < модификаторы команды >

Первым (и единственным обязательным) компонентом является *имя команды*, правила написания которого были рассмотрены выше.

Параметры команды – это, как правило, имена объектов (файлов и каталогов), над которыми выполняется операция. Количество, назначение и порядок записи параметров уникальны для каждой команды³³. Первый параметр отделяется от имени команды пробелом, если параметров несколько, они также должны быть разделены пробелами. Некоторые команды (например, команда VER) не имеют параметров, некоторые команды имеют параметры, но их значения могут задаваться «по умолчанию»: например, команда VOL, выполненная без параметра «имя тома», выведет метку активного тома, а команда DIR без параметра – выведет оглавление текущего каталога.

³² Формат строки «путь к файлу» подробно описан в п. 6.2.2.2 учебного пособия. Путь к текущему каталогу в командах не указывается – он известен операционной системе. Если путь начинается с символа «\» (обратная косая черта), он считается заданным от корневого каталога активного диска, если с символов «..» (две точки) – то путь задан от родительского каталога.

³³ Формат команды и полную инструкцию по ее применению можно вывести на экран путем выполнения этой команды без параметров с ключом «/?».

Модификаторы команды, называемы также *ключами*, используются для указания конкретных условий ее применения. Модификаторы могут располагаться в командной строке как перед, так и после параметров команды. В качестве разделителя модификаторов используется символ «/», который записывается перед очередным модификатором.

Состав модификаторов (так же, как и состав параметров) уникален для каждой команды, единственным исключением является модификатор «/?», присутствующий во всех внутренних и многих внешних командах: выполнение команды с таким модификатором приведет к выводу на экран подробной инструкции по применению этой команды.

1.3 Примеры использования команд

Команда **DIR** читает и выводит на экран оглавления каталогов. Единственным параметром этой команды является путь к каталогу.

- **DIR** (без параметров) – текущий каталог;
- **DIR** – корневой каталог;
- **DIR..** – родительский каталог;
- **DIR D:\DOC** – каталог **DOC**, подчиненный корневому каталогу;
- **DIR C:\ /p** – постраничный просмотр корневого каталога;
- **DIR C:\Windows /p/s** – постраничный просмотр каталога **\Windows** и всех подчиненных ему каталогов.
- **DIR *.exe** – просмотр части текущего каталога, содержащей только файлы с расширением **.exe**.

Команда **MD** (**M**ake **D**irectory) создает подчиненные каталоги.

- **MD ddd** – создание каталога, подчиненного текущему;
- **MD ..bbb** – создание каталога, подчиненного родительскому каталогу;
- **MD \first_d** – создание каталога, подчиненного корневому;
- **MD D:\first_d\next_dir** – создание каталога, подчиненного каталогу **first_d**.

Команда **CD** (**C**hange **D**irectory) - изменение текущего каталога.

- **CD ** – переход к корневому каталогу из текущего;
- **CD ..** – переход к родительскому каталогу из текущего;
- **CD next_dir** – переход к дочернему каталогу **next_dir**;
- **CD next_dir\ddd\qqq** – переход к каталогу **qqq** (указан путь к новому каталогу, начиная от текущего);

- **CD \First_dir\next_dir\ddd\qqq** – переход к каталогу **qqq** (указан путь к новому каталогу, начиная от корневого).

Команда COPY – копирование файлов. Команда использует два параметра: первый указывает спецификацию исходного файла (что и откуда копируется), а второй – спецификацию его копии (куда копируется и под каким именем). Команда может использоваться также для слияния нескольких файлов, печати и вывода на экран или создания (вводом с клавиатуры) текстового файла.

- **COPY E:\TXT\File_1.txt C:\TEXT\File_2.txt** – полноформатная команда копирования файлов, в результате выполнения которой в каталоге **TEXT** тома **C** будет создан файл **File_2.txt** – копия файла **File1.txt**, расположенного в каталоге **TXT** тома **E**;
- **COPY File_1.txt E:\TEXT\File_2.txt** – копирование файла под новым именем из текущего каталога в каталог **TEXT**, подчиненный корневому каталогу тома **E**;
- **COPY File1.txt D:** – копирование файла из текущего каталога в корневой каталог тома **D** (имя копии совпадает с именем копируемого файла);
- **COPY File1+File2+File3 File.All** – объединение (слияние) трех файлов в один файл **File.All** (все файлы – в текущем каталоге);
- **COPY File.txt PRN** – печать файла («копирование» на принтер);
- **COPY File.txt CON** – просмотр файла («копирование» на экран). Здесь в качестве параметра команды используется имя **CON** (сокращение от **Console**), зарезервированное операционной системой для стандартных устройств ввода-вывода. При *вводе* данных устройство **CON** – это клавиатура, при *выводе* – экран видеомонитора. Аналогичного результата можно достичь и командой **TYPE File.txt**.
- **COPY CON File.txt** – создание (копирование с клавиатуры) в текущем каталоге нового текстового файла **File.txt**. После выполнения такой команды можно вводить с клавиатуры произвольный текст. Для завершения процесса создания файла следует ввести специальный управляющий символ «конец файла» (CTRL+Z) и нажать клавишу Enter.

Команда PATH задает список путей к каталогам, содержащим файлы с исполнимыми программами, имена которых указаны в качестве имен внешних команд.

Команда использует единственный параметр – *список путей поиска*, разделенных символом «;» (точка с запятой). После выполнения команды ее параметр (строка со списком путей к целевым каталогам) помещается в специальный буфер ОЗУ и используется для последовательного поиска соответствующих каталогов в порядке их расположения в списке (слева направо).

Если во внешней команде путь к исполняемому файлу не указан, и этот файл отсутствует в текущем каталоге, поиск будет производиться в соответствии со списком путей, введенным командой **PATH** и сохраненном в соответствующем буфере.

Команда **PATH** с параметром «;» очищает буфер, то есть *отменяет* все ранее установленные *пути поиска* исполнимых файлов.

Команда **PATH**, введенная *без параметра*, выводит на экран текущее состояние буфера.

1.4 Использование групповых имен файлов

В приведенных выше примерах команд предполагалось, что одной командой выполняется соответствующая операция над одним файлом (или файлом-каталогом), имя которого указано в параметре команды. Использование в имени файла специальных шаблонов (подстановочных символов) позволяет одной командой произвести операцию над целой группой файлов.

Допускается использование в именах файлов двух типов шаблонов: символ «?» обозначает «один любой символ», а символ «*» (звездочка) – «любое количество любых символов» в имени или расширении файла (каталога).

Например, команда **TYPE *.txt** последовательно выведет на экран содержимое всех файлов текущего каталога, имеющих расширение **.txt**, а команда **DEL ?*DUM.*** удалит из текущего каталога все файлы с любыми расширениями и именами, состоящими из пяти символов, оканчивающимися на «DUM».

1.5 Перенаправление вывода

Многие команды выводят на экран основные результаты своей работы (например, команды **DIR** и **TYPE**) и/или диагностические сообщения (например, команды **COPY** и **FORMAT**). С помощью специальных символов «>» или «>>», записываемых в конце командной строки, можно перенаправить выводимую командой информацию на другие устройства или записать ее в текстовые файлы.

Например, команда **DIR D:\>PRN** выведет оглавление корневого каталога диска **D** на принтер, а команда **DIR D:\>dir.lst** запишет его в файл **dir.lst** текущего каталога. Если файл с именем **dir.lst** отсутствует, он будет создан в результате выполнения команды. Если файл уже существует, он будет замещен новым файлом с этим же именем.

Для добавления сообщений в существующий файл используется пара символов «>>», например, команда **DIR C:\>>dir.lst**, выполненная после приведенной выше команды, «допишет» оглавление каталога в конец файла **dir.lst**.

Фиктивное (не существующее) внешнее устройство с системным именем **NULL** используется для подавления вывода сообщений на экран. Например, при успешном выполнении команды **COPY File1.txt File2.txt>NULL** будет заблокирован вывод стандартного сообщения этой команды «Один файл скопирован» (при этом, естественно, выполнение команды заблокировано не будет).

Неразумное использование блокировки вывода сообщений команд может приводить к негативным последствиям, например, если в предыдущем примере файл **File1.txt** не будет найден в текущем каталоге, пользователь не получит соответствующего уведомления (типа «*File not found*») и будет надеяться, что копия файла создана.

Гораздо более неприятная ситуация может возникнуть, например, при массовом удалении файлов командой **DEL *.*>NULL**, так как процедура удаления файлов в этом случае сделает паузу и запросит у пользователя подтверждения выполнения операции «*Are you sure? Yes / No*». Неопытный пользователь, не увидев на экране этого сообщения, может долго ждать, но так и не дожидается завершения операции, так как не подозревает о необходимости своего участия в ее продолжении.

1.6 Практические задания

Задание Лаб1.1. Используя команду **PROMPT**, установите вид приглашения:

- текущие дата и время;
- версия Windows;
- собственная фамилия и номер группы;
- произвольный текст, заключенный в угловые скобки <>;
- восстановите стандартный вид приглашения (активный том и путь к текущему каталогу).

Задание Лаб1.2. Используя команду **MD**, создайте свой личный каталог с произвольным именем.

Задание Лаб1.3. Используя команду **COPY**, создайте в личном каталоге два коротких текстовых файла, содержащих по одной строке текста – Ваши фамилия, имя и отчество на русском и английском языках.

Задание Лаб1.4. Используя команду **REN**, переименуйте созданные файлы.

Задание Лаб1.5. Создайте в личном каталоге трехуровневую систему подчиненных каталогов.

Задание Лаб1.6. Скопируйте в каждый из созданных каталогов под различными именами файлы, созданные при выполнении задания 1.3.

Задание Лаб1.7. Скопируйте одной командой файл из одного подчиненного каталога в другой подчиненный каталог того же уровня.

Задание Лаб1.8. Создайте в личном каталоге новый файл путем объединения двух файлов, созданных при выполнении 3-го задания.

Задание Лаб1.9. Используя команды **COPY** и **TYPE**, просмотрите на экране содержимое всех файлов, созданных при выполнении выполненных заданий.

Задание Лаб1.10. Установите текущим один из созданных каталогов и сохраните его оглавление в файле **Direct.txt**, расположенном в этом каталоге.

Задание Лаб1.11. Сохраните оглавления личного каталога и всех подчиненных ему каталогов в файле **My_Dir.txt**, расположенном в личном каталоге. Предложите несколько вариантов формирования такого файла.

Задание Лаб1.12. Используя программную оболочку Norton Commander (или любой ее аналог):

- отредактируйте файлы, созданные при выполнении задания 3 (например, дополните текст Вашим домашним адресом).
- выполните задания с 2 по 6 без прямого использования командной строки.

Лабораторная работа №2. *Программирование пакетных файлов*

Задачи выполнения лабораторной работы – изучение расширенного набора команд и освоение техники программирования пакетных файлов.

Несколько команд могут быть построчно записаны в текстовый ASCII-файл, который должен иметь стандартное расширение **.bat** (от англ. *batch* – пакет, пачка). Такой файл является «исполнимым файлом», то есть имеет статус программы, которую можно выполнить, используя имя файла в качестве внешней команды. При этом каждая команда должна быть записана в отдельной строке файла, и интерпретатор будет последовательно выполнять содержащиеся в пакетном файле команды в порядке их записи, подобно тому, как если бы они были введены пользователем непосредственно в командной строке.

Пакетные файлы полезны тогда, когда необходимо выполнить часто повторяющуюся последовательность команд. Например, в результате выполнения пакетного файла `111.bat` (листинг 8.1) в текущем каталоге будет создан новый каталог `NewDir`, затем в этот каталог будет скопирован из текущего каталога файл `qqq.txt` под именем `ppp.txt`, затем каталог `NewDir` будет установлен текущим, и на экран будет выведен текст, содержащийся в файле `ppp.txt`.

```
MD NewDir
COPY qqq.txt NewDir\ppp.txt
CD NewDir
TYPE ppp.txt
```

Листинг 8.1 – Текст простейшего пакетного файла 111.bat

2.1 Использование переменных

В приведенном выше примере (листинг 8.1) параметры всех команд пакетного файла представлены константами, что позволяет усомниться в его полезности (так же, как и в полезности любой другой компьютерной программы, генерирующей одинаковые результаты при каждом ее выполнении: попытайтесь оценить пользу от калькулятора, который запрограммирован на выполнение единственной операции $2 + 3 = 5$).

Команды пакетного файла могут содержать имена переменных вместо своих фактических параметров, вместо модификаторов и даже вместо имен ко-

манд, что (вместе с наличием условных операторов и операторов циклов) позволяет считать пакетный файл «настоящей» компьютерной программой, написанной на языке командной строки.

Язык командной строки позволяет использовать в тексте пакетного файла не более 10 различных переменных, имена которых должны состоять из двух символов: первый символ – это всегда знак процента, а второй символ – одна из десятичных цифр. Таким образом, переменные оказываются «пронумерованными» в диапазоне от %0 до %9.

В языке командной строки *отсутствует оператор присваивания* – единственным способом присвоения переменным их фактических значений является передача этих значений через список параметров, которые записываются после имени пакетного файла при выполнении соответствующей внешней команды (подобно тому, как это делается при вызовах функций и процедур в языках программирования). Значение параметра списка (отделенного от соседних параметров символом «пробел») автоматически подставляется в текст пакетного файла вместо переменной с соответствующим номером. Например, переменная %2 получит перед выполнением файла значение второго по порядку параметра из списка, а переменная %5 – значение 5-го параметра.

```
MD %1
COPY %2 %1\%3
CD %1
TYPE %3
```

Листинг 8.2 – Текст пакетного файла 222.bat

Пакетный файл 222.bat (листинг 8.2) – функциональный аналог файла 111.bat, но он более универсален, так как содержит имена переменных вместо параметров команд.

Результаты выполнения команд 222.bat NewDir qqq.txt ppp.txt и 111.bat будут одинаковыми, так как в процессе интерпретации текста 222.bat (листинг 8.2) будут выполнены соответствующие подстановки: переменная %1 получит значение 1-го параметра NewDir, вместо переменной %2 будет подставлено значение 2-го параметра qqq.txt, а переменная %3 будет заменена на ppp.txt.

Используя переменные при программировании пакетных файлов, следует учитывать ряд специфических особенностей языка командной строки.

- 1) Все переменные имеют строковый тип данных, что не позволяет применять к ним арифметические операции.
- 2) Все переменные имеют статус локальных, область их «видимости» ограничена пакетным файлом, в котором они использованы. Одноименные переменные, используемые в «головном» и «подчиненном» (вызываемом из головного) пакетных файлах – это разные переменные.
- 3) Переменная %0 не соответствует никакому параметру – она получает значение имени пакетного файла (то есть той части командной строки, которая расположена левее первого пробела).
- 4) Соответствие имен переменных номерам параметров команды может быть изменено командой **SHIFT** в процессе исполнения пакетного файла: однократное выполнение этой команды сдвигает список «номеров» переменных на одну позицию вправо относительно списка введенных параметров (листинг 8.4).
- 5) Применение команды **SHIFT** также позволяет при выполнении пакетного файла использовать количество параметров, превышающее количество используемых в нем переменных (листинг 8.5).
- 6) Если список параметров команды оказался короче списка «номеров» переменных, использованных в тексте пакетного файла (с учетом п.5), все неопределенные переменные получают значение «пустая строка».

2.2 Специальные команды пакетных файлов

Специальные команды существенно повышают эффективность применения пакетных файлов, делая их полноценными программами. В таблице 8.2 приведены некоторые из таких команд с их кратким описанием. Для получения полного перечня команд с детальными инструкциями по их применению можно воспользоваться командой **HELP** или выполнить требуемую команду с модификатором «/?».

Таблица 8.2 - Специальные команды пакетных файлов

Команда	Выполняемая функция	Формат
ECHO	Вывод сообщений, блокировка отображения команд	ECHO <произвольный текст> ECHO OFF, ECHO ON
GOTO	Переход на метку	GOTO <метка строки>
IF	Условное выполнение команды	IF <условие> <команда_1> [ELSE < команда_2>]

Продолжение таблицы 8.2

Команда	Выполняемая функция		Формат
FOR	Циклическое выполнение команды	FOR %%<переменная цикла> IN <список значений>	DO <команда>
CALL	Вызов подчиненного bat-файла	CALL <имя.bat> <параметры>	
SHIFT	Сдвиг списка фактических параметров bat-файла относительно списка используемых переменных		SHIFT
REM	Записывается в начале строки, блокирует выполнение команды. Обычно используется для записи комментариев в тексте пакетного файла или для временного блокирования команд		REM <текст>
:	Записывается в начале строки (не более 8 символов), присваивает этой строке статус метки, которая может использоваться в командах GOTO		:<текст>
@	Записывается в начале строки, подавляет отображение этой строки на экране при выполнении пакетного файла		@ <команда>

Листинги 8.3 – 8.7 иллюстрируют возможности специальных команд.

При выполнении пакетного файла, представленного на листинге 8.3, оглавления каталога **TEXT** в каждом из трех его состояний будут последовательно записаны в файл **Dir.lst**, расположенный в текущем каталоге. Содержимое этого файла затем будет выведено на экран командой **TYPE**. Стандартные сообщения команд **COPY** и **DEL** выводиться не будут – вместо них на экран будет выводиться текст, указанный в параметрах команд **ECHO**.

```

@ECHO OFF
MD TEXT
DIR TEXT >Dir.lst
ECHO Оглавления всех каталогов - в файле Dir.lst
ECHO Копирование текстовых файлов из текущего каталога
COPY *.TXT TEXT\*.* > NULL
ECHO Копирование завершено
DIR TEXT >>Dir.lst
ECHO Удаление текстовых файлов из текущего каталога
DEL TEXT\*.TXT > NULL
ECHO Удаление завершено
DIR TEXT >>Dir.lst
ECHO Просмотр оглавлений каталогов
TYPE Dir.lst

```

Листинг 8.3 – Примеры использования команды ECHO

Команда Shift (листинг 8.4) сдвигает список параметров пакетного файла на одну позицию влево относительно списка переменных. Первая команда

ECHO выведет на экран значение первого параметра, а вторая (точно такая же) – значение второго параметра.

```
ECHO %1  
Shift  
ECHO %1
```

*Листинг 8.4 – Пример использования команды **Shift***

Листинг 8.5 иллюстрирует возможность использования практически неограниченного количества параметров при ограниченном количестве переменных. В этом примере единственная переменная **%1** последовательно получает значения всех параметров, начиная с первого, и каждое ее значение выводится на экран до тех пор, пока список параметров не будет исчерпан (т.е пока переменная **%1** не получит значения «пусто»).

```
:Loop  
ECHO %1  
SHIFT  
IF -%1==- GOTO Exit  
GOTO Loop  
:Exit  
ECHO Список параметров исчерпан
```

*Листинг 8.5 – Пример использования команд **IF** и **Shift***

Листинг 8.6 представляет команду **FOR**, обеспечивающую циклическое выполнение других команд. Переменные цикла **%%a**, **%%b**, **%%c** и **%%d** последовательно получают значения из списка, заданного в разделе **IN**, и используются в качестве параметров команд, заданных в разделе **DO**.

```
FOR %%a IN (Dir1 Dir2 Dir3) DO MD %%a  
FOR %%b IN (Dir1 Dir2 Dir3) DO COPY File1 %%b\  
FOR %%c IN (*.%1) DO ECHO %%c  
FOR %%d IN (txt doc xls) DO IF exist %1.%%d COPY %1.%%d DIR1\
```

*Листинг 8.6 – Примеры использования команды **FOR***

Листинг 8.7 иллюстрирует использование команды **CALL**, с помощью которой из головного пакетного файла (**QQQ.bat**) вызывается подчиненный

(**PPP.bat**) с передачей ему четырех параметров, два из которых – это локальные переменные головного файла, а два других – текстовые константы.

Файл **QQQ.bat**

```
@ECHO OFF
COPY %2 %1\%3
CALL PPP.bat File_1 %3 %1 File_2
TYPE %1\File_2
```

Файл **PPP.bat**

```
@ECHO OFF
ECHO Исходные файлы %1 и %2
CD %3
COPY %1+%2 %4 >Null
ECHO Слияние файлов завершено
```

*Листинг 8.7 – Пример использования команды **CALL***

Пусть оба файла размещены в одном каталоге. При выполнении команды **QQQ.bat Dir1 Name1 Name2** переменные %1, %2 и %3 головного файла **QQQ.bat** получают значения соответственно **Dir1**, **Name1** и **Name2**.

Вторая команда этого файла скопирует файл **Name1** под именем **Name2** в подкаталог **Dir1**. Следующая команда **CALL** выполнит файл **PPP.bat**, передав ему в качестве первого параметра текстовую константу **File_1**, в качестве второго параметра – значение переменной %3 головного файла (то есть **Name2**), в качестве третьего параметра – значение первого параметра головного файла (то есть **Dir1**), а в качестве четвертого параметра – текстовую константу **File_2**.

В результате переменные %1, %2, %3 и %4 подчиненного файла **PPP.bat** получают значения соответственно **File_1**, **Name2**, **Dir1** и **File_2**, команда **COPY** соединит два файла **File_1** и **Name2** и сохранит результат соединения в файле **File_2**.

После завершения работы подчиненного пакетного файла **PPP.bat** будет выполнена команда **TYPE** головного файла, которая выведет на экран содержимое файла **File_2**.

Можно вызвать подчиненный пакетный файл из головного и без команды **CALL** (вместо строки **CALL PPP.bat File_1 %3 %1 File_2** записать

строку **PPP.bat File_1 %3 %1 File_2**). Однако, в этом случае после завершения работы подчиненного файла не произойдет возврата к головному файлу, и вся программа завершит работу последней командой подчиненного файла.

2.3 Практические задания

Задание Лаб2.1. Подготовьте пакетные файлы, примеры которых приведены на листингах 8.1 – 8.7. Протестируйте их на различных наборах входных параметров, оцените работоспособность этих пакетных файлов.

Задание Лаб2.2. Используя команды **Help** и **IF/?**, изучите синтаксис команды **IF...ELSE** и подготовьте несколько примеров использования этой команды при программировании пакетных файлов. Прокомментируйте 4-ю строку пакетного файла, представленного на листинге 8.5. Можно ли заменить эту строку на **IF +%1==+ GOTO Exit** или на **IF %1== GOTO Exit** без потери работоспособности программы? Ответ подтвердите результатами эксперимента.

Задание Лаб2.3. Подготовьте пакетный файл **self_cloning.bat**, при выполнении которого в текущем каталоге создается его копия под новым именем (имя указывается параметром команды). При этом созданная копия файла должна оставаться способной к само-клонированию – то есть к созданию своих собственных копий под разными именами.

Задание Лаб2.4. Оцените возможности пакетного файла **Universal.bat**, текст которого содержит единственную команду: **%1 %2 %3**. Как применить такой пакетный файл для копирования и переименования файла, а также для удаления группы файлов заданного каталога? Попытайтесь найти и другие применения такому универсальному пакетному файлу.

Лабораторная работа №3.

Файловая система ПК

Рассмотренный выше язык командной строки используется на верхнем уровне файловой системы и базируется на пользовательской модели дискового пространства ПК, в основе которой – множество иерархических («древовидных») структур вида «том – каталоги – файлы». Исполнение команд осуществляется соответствующими системными функциями, алгоритмы работы которых являются объектом исследования в лабораторной работе №3.

Основная задача выполнения лабораторной работы – экспериментальное исследование модели данных, поддерживаемой файловыми FAT-системами (разделы 6.1 и 6.2.1 учебного пособия), по результатам которого требуется подтвердить, опровергнуть или уточнить описания этих алгоритмов (раздел 6.2.2).

3.1 Инструментальное программное обеспечение

1. Виртуальная DOS-машина.
2. Norton Commander (DOS Navigator).
3. Программа-анализатор дискового пространства DiskEdit.exe.
4. Электронный справочник HELP (файлы Help.exe и Help.dat).

3.2 Практические задания

Задание Лаб3.1. **Исследование структуры таблицы разделов диска**

Прямым доступом к таблице разделов (таблица 6.1) определите и сохраните в отчете следующие параметры для каждого из разделов:

- аппаратные адреса;
- тип ОС и разрядность FAT;
- количество секторов.

Задание Лаб3.2. **Исследование структуры загрузочного сектора тома**

Просмотрите содержимое boot-сектора (таблица 6.2) одного из томов «виртуального» ПК. Определите и сохраните в отчете:

- параметры физического форматирования тома (количество рабочих поверхностей, цилиндров, секторов на дорожке, размер кластера);
- количество секторов, выделенных FAT и корневому каталогу;
- размер (в байтах) таблицы параметров форматирования диска;
- машинный код программы загрузки ОС.

Задание Лаб3.3. Исследование структуры корневого каталога

Просмотрите (командой DIR и программой Diskedit) оглавление корневого каталога одного из томов. Определите и сохраните в отчете следующие параметры этого каталога:

- размер каталога (в секторах и байтах);
- физическое расположение (номера занятых каталогом секторов);
- размер регистрационной записи о дочернем объекте каталога;
- количество дочерних объектов корневого каталога, в том числе: файлов, подкаталогов, системных и скрытых файлов, меток томов;
- количество удаленных файлов.

Задание Лаб3.4. Исследование структуры байта атрибутов файла

- С помощью программы Diskedit определите адрес (смещение в регистрационной записи о дочернем объекте) байта атрибутов файла и порядковые номера битов для каждого атрибута.
- Определите системные, скрытые и защищенные от модификации файлы корневого каталога, а также файлы, требующие архивирования при создании очередной резервной копии тома.
- Установите для одного из текстовых файлов атрибут «только чтение» и экспериментально подтвердите повышение уровня защищенности файла.
- Установите для одного из текстовых файлов атрибут «является каталогом» и оцените последствия такого изменения.
- Измените метку тома стандартными средствами (командой *Label*). Измените метку тома с помощью программы Diskedit и оцените результат с помощью команды Vol. Предложите способ сокрытия от (неквалифицированного) пользователя факта наличия на диске большой группы файлов.

Задание Лаб3.5. Исследование структуры таблицы FAT

Прямым доступом к области FAT определите и сохраните в отчете:

- номера секторов, занятых каждой копией FAT;
- количество объектов (файлов и подчиненных каталогов) на томе;
- общее количество кластеров на томе, в том числе – свободных, занятых и «сбойных» кластеров;
- объем (в байтах) свободного пространства на томе (подтвердите ответ командой *Ctrl+L Norton Commander'a*).

Задание Лаб3.6. Исследование алгоритма создания подчиненных каталогов

- Создайте (командой *MD*) подчиненный каталог и каталог, подчиненный этому подчиненному каталогу; просмотрите оглавления созданных каталогов командой *DIR* и программой *Diskedit*.
- Определите и сохраните в отчете следующие параметры для каждого из двух созданных каталогов:
 - размер и начальные адреса (номера начальных кластеров);
 - начальные адреса дочерних объектов и родительских каталогов.
- Скопируйте несколько файлов в любой из подчиненных каталогов, созданных при выполнении предыдущего задания.
 - изменился ли размер (в кластерах) этого каталога?
 - сколько всего файлов можно зарегистрировать в этом каталоге, чтобы сохранился его начальный (минимальный) размер?
 - ответ на последний вопрос подтвердите экспериментально.

Задание Лаб3.7. Исследование алгоритмов удаления и восстановления файлов

- Просмотрите содержимое кластеров, занятых одним из файлов, скопированных в каталог при выполнении предыдущего задания.
- Удалите (командой *DEL*) этот файл, повторно просмотрите содержимое этих же кластеров, прокомментируйте результат. Что изменилось в таблице FAT и в родительском каталоге удаленного файла?
- Оцените возможность восстановления удаленного файла и восстановите его с помощью программы *Diskedit*.
- Удалите (командой *DEL*) группу из нескольких файлов, оцените возможность восстановления удаленной группы файлов, попытайтесь восстановить эти файлы с помощью программы *Diskedit*.

Задание Лаб3.8. Исследование алгоритмов перемещения файлов

- Выберите по своему усмотрению два каталога *одного тома*, просмотрите их оглавления командой *DIR* и программой *Diskedit*.
- Используя команду *F6* Norton Commander'a, переместите файл из одного из этих каталогов в другой. Что изменилось в каталогах и таблице FAT?
- Выполните операцию перемещения файла между каталогами *разных томов* и сравните результаты этой операции с предыдущей.
- Опишите алгоритмы перемещения файла между каталогами:
 - a) каталоги размещены на одном томе;
 - b) каталоги размещены на разных томах.

Лабораторная работа №4.

Исследование адресного пространства ПК

Основная задача выполнения лабораторной работы – освоение инструментальных средств (программ – анализаторов памяти) в процессе исследования системных областей памяти ПК – таких, как таблица векторов прерываний и область данных BIOS.

Сегментная организация базового адресного пространства ПК, его типовая структура и процедура обработки прерываний рассмотрены в разделах 4.6.1, 4.6.2 и 7.1.3 учебного пособия – изучение всех этих разделов должно предшествовать выполнению 4-й лабораторной работы.

4.1 Инструментальное программное обеспечение

1. Виртуальная DOS-машина.
2. Электронный справочник HELP (файлы Help.exe и Help.dat).
3. Программы-анализаторы памяти PEEK.com и Fx_Show.com.
4. Универсальный калькулятор Calc.com. Рекомендуется загружать калькулятор до загрузки анализатора памяти PEEK.com – тогда калькулятор будет доступен непосредственно из PEEK.com (F9).

4.2 Практические задания

Задание Лаб4.1. ***Исследование структуры таблицы векторов прерываний***

- Определите начальный и конечный адреса таблицы векторов прерываний. Сколько векторов прерываний может содержать эта структура данных?
- Используя справочник HELP, определите (и сохраните в отчете) назначение программ обработки прерываний №8, №9, №10_h, №16_h и №21_h.
- Какие из этих прерываний аппаратные, а какие – программные?
- Прямым доступом к таблице векторов прерываний определите начальные адреса программ обработки перечисленных выше прерываний.
- Определите состав программ, загруженных в память ПК и номера прерываний, «перехваченных» этими программами.
- Определите (и сохраните в отчете) назначение прерываний №1F_h и №43_h и начальные адреса областей памяти, связанных с этими прерываниями.
- Определите начальный адрес программы PEEK.com, просмотрите машинный код этой программы.
- Измените первый байт первой машинной команды программы PEEK.com, прокомментируйте полученный результат.

Задание Лаб4.2. *Исследование структуры области данных BIOS*

- Используя справочник HELP, определите начальный и конечный адреса области данных BIOS. Какая информация хранится в ячейках памяти с этими адресами?
- Определите состав параллельных и последовательных адаптеров (LPT* и COM*), установленных в Вашем «виртуальном» ПК, и базовые адреса этих адаптеров. Запишите (и сохраните в отчете) базовые адреса в сегментной и линейной формах в двоичной и шестнадцатеричной системах счисления.
- Определите содержимое старшего и младшего байтов двухбайтового машинного слова, расположенного по адресу [0040:001C]_h. Запишите это машинное слово в шестнадцатеричной и двоичной системах счисления.
- Повторите предыдущее задание для машинного слова, расположенного по адресу [0000:041C]_h. Сравните и прокомментируйте результаты.
- Нажимая (не менее 16 раз) на любую из символьных клавиш клавиатуры, проследите за изменениями машинного слова по адресу [0040:001C]_h. Какая информация содержится в этом машинном слове?

Лабораторная работа №5.

Клавиатура ПК

В процессе выполнения 5-й лабораторной работы продолжается экспериментальное исследование области данных BIOS в той ее части, которая используется обработчиком прерывания №9 и прикладными программами в процессе ввода данных с клавиатуры. Исследуемые структуры данных представлены в таблицах 7.1 и 7.2, блок-схемы и иллюстрации исследуемых алгоритмов приведены на рисунках 7.3 – 7.9.

При проведении экспериментов используются те же инструментальные средства, как и при выполнении предыдущей лабораторной работы.

5.1 Практические задания

Задание Лаб5.1. Исследуйте процесс изменения состояния «флаговых байтов» в процессе манипулирования управляющими клавишами Shift, Ctrl, CapsLock, NumLock. По результатам эксперимента составьте таблицу, аналогичную таблице 7.2. Результаты представьте в шестнадцатеричном и двоичном форматах и сохраните в отчете.

Задание Лаб5.2. Установите режим «верхний регистр», не нажимая клавиши CapsLock. Попробуйте отменить установленный режим с помощью этой клавиши. Проведите аналогичный эксперимент с клавишей NumLock. Прокомментируйте результаты эксперимента.

Задание Лаб5.3. Запишите адреса ячеек буфера клавиатуры, в которые будут записаны и из которых будут прочитаны Scan-код и ASCII-код очередной символьной клавиши. Экспериментально подтвердите правильность своих предположений.

Задание Лаб5.4. Введите символы G , g , Π и n , связанные с одной клавишей. Определите Scan-код этой клавиши и ASCII-коды введенных символов. Результаты представьте в шестнадцатеричной, десятичной и двоичной системах счисления. Выполните аналогичные действия для другой символьной клавиши. Результаты сохраните в отчете.

Задание Лаб5.5. Объясните, почему обработчик 9-го прерывания сгенерировал 4 различных ASCII-кода символа по одному значению Scan-кода клавиши.

Задание Лаб5.6. Введите символы, указанные в задании №4, прямым набором их ASCII-кодов на цифровой клавиатуре (при нажатой клавише Alt). Зафиксируйте промежуточные состояния ячейки с адресом $[0040:0019]_h$ после ввода каждой очередной цифры ASCII-кода символа, а также состояние этой ячейки и соответствующих ячеек буфера клавиатуры после отпускания клавиши Alt. Опишите алгоритм ввода символа прямым набором его ASCII-кода.

Задание Лаб5.7. Введите прямым набором символ с ASCII-кодом «1234». Прокомментируйте результат ввода.

Задание Лаб5.8. Введите прямым набором «управляющие символы» (ASCII-коды от 0 до 31). Используя электронный справочник HELP, определите комбинации клавиш (Ctrl+клавиша), используемых для ввода управляющих символов.

Задание Лаб5.9. Используя электронный справочник HELP и технологию прямого набора кодов символов, составьте таблицу ASCII-кодов символов псевдографики.

Лабораторная работа №6.

Видеосистема ПК

В процессе выполнения лабораторной работы исследуется область данных BIOS (параметры видеосистемы), система кодирования видеоданных и структура видеопамати в текстовых режимах работы видеоадаптера, структура таблиц знакогенераторов.

Справочная информация о структурах данных, обслуживающих процесс вывода данных на видеомонитор ПК приведена в таблицах 7.4 и 7.6, система кодирования видеоданных описана в п.7.3.4.1, структура таблиц знакогенераторов рассматривается в п. 7.3.5 учебного пособия.

При проведении экспериментов используются те же инструментальные средства, как и при выполнении предыдущей лабораторной работы.

6.1 Практические задания

Задание Лаб 6.1. Используя информацию области данных BIOS, определите и сохраните в отчете следующие параметры видеосистемы Вашего «виртуального» ПК:

- тип видеоадаптера;
- объем видеопамати;
- номер текущего видеорежима;
- число символов в строке;
- размер видеостраницы (в байтах);
- начальный адрес активной видеостраницы;
- координаты курсора для каждой из видеостраниц текущего текстового видеорежима.

Задание Лаб 6.2. Отобразите на экране область данных BIOS, установите режим редактирования памяти (F5 в программе Peek.com).

- как изменилась форма курсора?
- что (и по какой причине) изменилось в области данных BIOS?
- перемещая курсор клавишами-стрелками, наблюдайте за изменением координат курсора в каждой из видеостраниц.
- какая программа вносит изменения в эти ячейки памяти?
- какая программа «рисует» на экране курсор «правильной» формы и в «правильном» месте экрана?

Задание Лаб 6.3. Выведите на экран образ активной видеостраницы. Прокомментируйте результаты отображения и объясните систему кодирования видеоданных в текстовом режиме.

Задание Лаб 6.4. Определите адрес машинного слова (байт символа и байт атрибутов), описывающего знакоместо экрана с заданными координатами (x,y) на активной видеостранице. Прочитайте (и сохраните в отчете) машинное слово по этому адресу, убедитесь в том, что видеоадаптер отображает на экране символ, соответствующий прочитанному ASCII-коду и значению атрибута (цвета) символа.

Задание Лаб 6.5. Используя векторы прерываний №1F_h и №43_h, определите расположение соответствующих таблиц знакогенератора.

Задание Лаб 6.6. Исследуйте структуру одной из таблиц знакогенератора. По данным таблицы составьте битовую матрицу, описывающую символ с заданным ASCII-кодом (пример – на рисунке 7.11).

Приложение А.

Инструкция по установке виртуальной DOS-машины

1. Скопируйте папку «*Эмулятор MS DOS*» со всем ее содержимым (установочный файл виртуальной машины **VirtualBox-5.2.18-124319-Win.exe**, файлы образов жесткого и гибкого дисков **dos.ova** и **floppy.img** и файл с текстом этой инструкции) на рабочий ПК.
2. Запустите установочный файл **VirtualBox** и дождитесь завершения установки виртуальной машины Oracle VM VirtualBox на Ваш рабочий ПК (в процессе установки все опции принимайте по умолчанию, на все вопросы установщика отвечайте утвердительно: *Next, Yes*).
3. После завершения установки создайте на рабочем столе (или в другом удобном Вам месте) ярлык приложения Oracle VM VirtualBox и запустите это приложение.
4. В меню «*Файл*» выберите «*Импорт*», укажите путь к файлу **dos.ova** и выберите этот файл. В результате будет импортирован образ жесткого диска (C:\), на котором уже будут установлены инструментальные средства, необходимые для выполнения лабораторных работ.
5. В меню «*Настройки*» выберите пункт «*Носители*», укажите путь к файлу **floppy.img** и выполните. В результате дополнительно будет подключен образ гибкого диска (A:\).

DOS-машина на Вашем ПК готова к работе.

Учебное издание

Волк Владимир Константинович

ИНФОРМАТИКА
ВВОДНЫЙ КУРС
ДЛЯ СТУДЕНТОВ ИТ-СПЕЦИАЛЬНОСТЕЙ

Учебное пособие

В авторской редакции

Библиотечно-издательский центр КГУ.
640020, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.