Воронежский институт высоких технологий

Российский Новый Университет
(Воронежский филиал)

И.Я. Львович,
Ю.П. Преображенский,
В.В. Ермолова

ОСНОВЫ ИНФОРМАТИКИ

Учебное пособие

Воронеж 2014

УДК
ББК

Основы информатики: Учеб. пособие /И.Я. Львович, Ю.П. Преображенский, В.В. Ермолова. ВИВТ, Воронеж, 2014, 253 с.

В учебном пособии в рамках Федеральных государственных образовательных стандартов рассматриваются понятия информатики, информационных процессов систем и технологий. Подробно рассмотрены математические основы цифровых автоматов, кодирования; алгоритмические основы программирования и управления. Большое внимание уделено прикладным вопросам информатики, сетевым информационным технологиям.

Предназначено для студентов, обучающихся по направлениям подготовки «Информационные системы и технологии» и «Информатика и вычислительная техника», а также для магистров и аспирантов.

Табл. 19, Ил. 87., Библиогр. 20 назв.

Печатается по решению Учебно-методического совета Воронежского института высоких технологий.

Научный редактор:
Заслуженный деятель науки РФ, д-р техн. наук, профессор, академик РАЕН Я.Е. Львович

Рецензенты:
д-р техн. наук , проф. каф. САПРИС ВГТУ Юрочкин А.Г.
д-р техн. наук, проф., зав.каф. ТАСЭМ ВГТУ Чопоров О.Н.

ISBN

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В учебных планах подготовки бакалавров по направлениям «Информатика и вычислительная техника» и "Информационные системы и технологии" дисциплина «Информатика» входит в состав базовой части математического и естественнонаучного цикла дисциплин. Она создает теоретическую основу для изложения и понимания таких курсов, как «Теория информационных процессов и систем», "Информационные технологии", "Архитектура информационных систем", «Управление данными», «Технологии программирования» и других фундаментальных дисциплин технического профиля.

Целями данного учебного пособия являются:

1) осуществить четкое изложение основных понятий и современных подходов к информатике как самостоятельной науке естественнонаучного направления;

2) на основании анализа объекта и предметной области информатики изложить фундаментальные категории и аксиомы, которые служат стержнем информатики как науки;

3) дать математические основы информатики как основу, инструмент для решения прикладных задач;

4) показать студенту возможность построения «дерева» информатики, базируясь на триадах «информация – информационные технологии – информационный ресурс» и «модель – алгоритм – программа»;

5) заложить первоначальные основы знаний студентам первого курса по структуре и функциям блоков ЭВМ, алгоритмизации и программированию.

1. ПОНЯТИЕ ИНФОРМАЦИИ И ПОДХОДЫ К ЕЕ КОЛИЧЕСТВЕННОЙ ОЦЕНКЕ

1.1 Понятие и виды информации

Термин "информация" происходит от латинского слова "Informatio" – разъяснение, изложение, осведомленность. Можно считать, что этот термин в начальном представлении является общим понятием, означающим некоторые сведения, совокупность данных, знаний и т.д. Понятие информации должно быть связано с определенным объектом, свойства которого она отражает. Кроме того, наблюдается относительная независимость информации от ее носителя, поскольку возможны ее преобразование и передача по различным физическим средам с помощью разнообразных физических сигналов безотносительно к ее содержанию, т.е. к семантике, что и явилось центральным вопросом многих исследований, в том числе и в философской науке. Информация о любом материальном объекте может быть получена путем наблюдения, натурного либо вычислительного эксперимента, а также на основе логического вывода. Поэтому говорят о доопытной (или априорной) информации и послеопытной (т.е. апостериорной), полученной в итоге эксперимента.

Для человека любое восприятие реальных объектов окружающей действительности происходит через ощущения. Органы чувств человека и высшая нервная система позволяют ему воспринимать объекты. При обмене информацией существуют источник в виде объекта материального мира и приемник - человек либо какой-то материальный объект. Информация возникает за счет отражения, которое является свойством всей материи, любой материальной системы. Свойство отражения совершенствуется по мере развития материи от элементарного отражения до высшей его формы – сознания. Процесс отражения означает взаимодействие объектов материального мира. Этот процесс наиболее прост в неорганической природе. Здесь преобладают механические, химические и физические взаимодействия. При таком отражении объекты пассивны. Новые формы отражения (физиологическое и психологическое) возникают в органической природе. В живом организме на основе отражения формируется способность приспосабливаться к изменяющимся окружающим условиям. У человека получают развитие более сложные формы отражения: познавательная и творческая. Эти формы носят сознательный характер и позволяют человеку активно воздействовать на окружающий мир.

Рассмотрим некоторую классификацию информации. Для того, чтобы она была полезной, она должна быть основана на некоторой системе. Обычно при классификации объектов одной природы в качестве базы для классификации используется то или иное свойство (или их набор) объектов.

Как правило, свойства объектов можно разделить на два больших класса: внешние и внутренние свойства.

Внутренние свойства - это свойства, органически присущие объекту.

Они обычно «скрыты» от изучающего объект и проявляют себя косвенным образом при взаимодействии данного объекта с другими.

Внешние свойства - это свойства, характеризующие поведение объекта при взаимодействии с другими объектами.

Поясним сказанное на примере. Масса является внутренним свойством вещества (материи). Проявляет же она себя во взаимодействии или в ходе некоторого процесса. Отсюда появляются такие понятия физики, как гравитационная масса и инерциальная масса, которые можно было бы назвать внешними свойствами вещества.

Подобное разделение свойств можно привести и для информации. Для любой информации можно указать три объекта взаимодействия: источник информации, приемник информации (ее потребитель) и объект или явление, которые данная информация отражает. Поэтому можно выделить три группы внешних свойств, важнейшими из которых являются свойства информации с точки зрения потребителя.

Качество информации - обобщенная положительная характеристика информации, отражающая степень ее полезности для пользователя.

Показатель качества - одно из важных положительных свойств информации (с позиции потребителя). Любое отрицательное свойство может быть заменено обратным ему, положительным.

Чаще всего рассматривают показатели качества, которые можно выразить числом, и такие показатели являются *количественными* характеристиками положительных свойств информации.

Обзор приведенных ситуаций позволяет сформулировать следующие определения свойств информации.

Релевантность - способность информации соответствовать нуждам (запросам) потребителя.

Полнота - свойство информации исчерпывающе (для данного потребителя) характеризовать отображаемый объект и / или процесс.

Своевременность (актуальность) - способность информации соответствовать нуждам потребителя в нужный момент времени.

Достоверность - свойство информации не иметь скрытых ошибок.

Доступность - свойство информации, характеризующее возможность ее получения данным потребителем.

Защищенность - свойство, характеризующее невозможность несанкционированного использования или изменения.

Эргономичность - свойство, характеризующее удобство формы или объема информации с точки зрения данного потребителя.

Адекватность - свойство информации однозначно соответствовать отображаемому объекту или явлению. Адекватность оказывается для потребителя внутренним свойством информации, проявляющем себя через релевантность и достоверность.

Выделяют следующие аспекты информации:

- прагматический,
- семантический,

- синтаксический.

Прагматический аспект связан с возможностью достижения поставленной цели с использованием получаемой информации. Этот аспект информации влияет на поведение потребителя. Если информация была эффективной, то поведение потребителя меняется в желаемом направлении, т. е. информация имеет прагматическое содержание. Таким образом, этот аспект характеризует поведенческую сторону проблемы.

Семантический аспект позволяет оценить смысл передаваемой информации, соотнося ее с информацией, хранящейся до появления данной. Семантические связи между словами или другими смысловыми элементами языка отражают словарь – *тезаурус*. Он состоит из двух частей: списка слов и устойчивых словосочетаний, которые сгруппированы по смыслу, и некоторого ключа, т е. алфавитного словаря, позволяющего расположить слова и словосочетания в определенном порядке. Тезаурус имеет особое значение в системах хранения информации, в которые могут вводиться семантические отношения, в основном подчинения, что позволяет на логическом уровне осуществлять организацию информации в виде отдельных записей, массивов и их комплексов. Существуют развитые тезаурусы, в которые включаются сложные высказывания и семантические связи между ними. Это позволяет хранить более сложную информацию и детально оценивать семантическое содержание вновь поступающей информации. Наличие тезауруса позволяет переводить поступающую семантическую информацию на некоторый стандартизированный семантический язык в соответствии с выбранным тезаурусом. Таким образом, при возникновении информации можно изменить исходный тезаурус. Степень изменения тезауруса может быть принята как характеристика количества информации.

Пример тезауруса

Приведем пример русско-английского тезауруса предметной области «Автоматический оптический контроль печатных плат». Особенность данной предметной области состоит в том, что она находится на стыке двух дисциплин: технологии производства печатных плат и машинного зрения, соответственно – термины «происходят» из разных терминосистем. Основным методом составления словаря была интроспекция, в качестве вспомогательного материала были использованы статьи, отечественные стандарты, словари по машинному зрению и компьютерной графике, словарь по технологии изготовления печатных плат.

В тезаурус вошло около 200 концепций, 800 английских и русских терминов и 700 однонаправленных связей между концепциями. В тезаурусе использованы следующие типы связей: *род – вид, часть – целое, следует – предшествует, используется для – использует, носитель свойства – свойство* (асимметричные связи), а также *ассоциация, коррелят* (симметричные связи). «Концентраторами» (*hubs*) тезаурусной сети являются

концепции *дефект* (26 смежных концепций), *печатная плата* (23 смежные концепции), *автоматический оптический контроль* (17 смежных концепций). На рисунке 1.1 представлен фрагмент тезаурусной сети (чтобы не перегружать рисунок, для каждой пары отображена только одна связь).

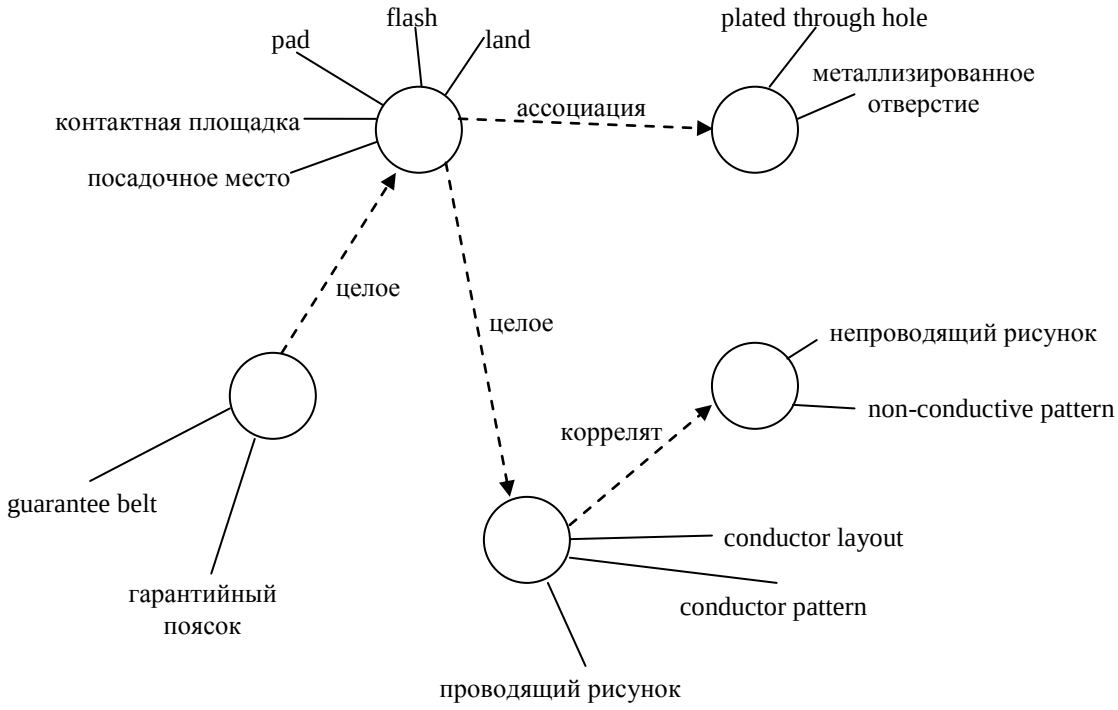


Рисунок 1.1 – Фрагмент тезаурусной сети

Синтаксический аспект информации связан со способом ее представления. В зависимости от реального процесса, в котором участвует информация, т.е. осуществляется ее сбор, передача, преобразование, отображение, представление, ввод или вывод, она представляется в виде специальных знаков, символов. Характерным носителем информации является сообщение, под которым обычно понимают все то, что подлежит передаче. Сообщения представляют в виде электрического сигнала, передаваемого по выбранной физической среде. Для этого сообщение подвергают преобразованию, т. е. придают ему электрический характер, далее кодированию, при котором сообщение превращается в некоторую, последовательность символов, однозначно его отображающих, и модуляции, при которой каждый элемент кода (либо код в целом) переводится в электрический сигнал, способный передаваться на заданное расстояние по выбранному каналу связи. Процессы преобразования, кодирования и модуляции исключительно многообразны, а синтаксический аспект информации при ее передаче в настоящее время хорошо развит. Иной характер синтаксический аспект имеет, например, при хранении информации. В этом случае могут быть предложены такие формы, при которых удается осуществить быстрый поиск, введение новой информации, вывод требуемой информации из информационной базы и в целом обновления базы данных. Требуемому представлению информации при ее

хранении отвечают разработанные к настоящему времени типовые структуры баз и банков данных, которые позволяют наилучшим образом реализовать информационное обслуживание пользователей в системе управления. Таким образом, развитие общества привело к тому, что оказалось необходимым хранить, обрабатывать, передавать, преобразовывать огромные объемы данных.

Виды информации

Все виды деятельности человека по преобразованию природы и общества сопровождались получением новой информации. Логическая, адекватно отображающая объективные закономерности природы, общества и мышления получила название *научной информации*. Ее делят по областям получения или пользования на следующие виды: *политическую, техническую, биологическую, химическую, физическую* и т.д.; по назначению – на *массовую* и *специальную*. Часть информации, которая занесена на бумажный носитель, получила название *документальной информации*. Любое производство при функционировании требует перемещения документов, т.е. возникает *документооборот*. Для автоматизированных систем управления информация в документах составляет *внешнее информационное обеспечение*. В то же время большая часть информации хранится в памяти ЭВМ на магнитных и оптических дисках и т.д. Она определяется как *внутримашинное информационное обеспечение*.

Наряду с научной информацией в сфере техники при решении производственных задач используется *техническая информация*. Она сопровождает разработку новых изделий, материалов, конструкций агрегатов, технологических процессов. Научную и техническую информацию объединяют термином *научно-техническая информация*; в сфере материального производства может циркулировать *технологическая информация*, закрепленная в конструкторско-технологической документации. В плановых расчетах существует *планово-экономическая информация*, которая содержит интегральные сведения о ходе производства, значения различных экономических показателей.

Информация с точки зрения ее возникновения и совершенствования проходит следующий путь: человек наблюдает некоторый факт окружающей действительности, это факт отражается в виде совокупности данных, при последующем структурировании в соответствии с конкретной предметной областью данные превращаются в знания. Таким образом, верхним уровнем информации как результата отражения окружающей действительности (результата мышления) являются *знания*. Знания возникают как итог теоретической и практической деятельности. Информация в виде знаний отличается высокой структуризацией. Это позволяет выделить полезную информацию при анализе окружающих нас физических, химических и прочих процессов и явлений. На основе структуризации информации формируется информационная модель объекта. По мере развития общества

информация как совокупность научно-технических данных и знаний превращается в базу системы информационного обслуживания научно-технической деятельности общества.

В настоящее время информация используется всеми отраслями народного хозяйства и, наряду с энергией, полезными ископаемыми, является ресурсом общества. С развитием общества возникает необходимость целесообразной организации *информационного ресурса*, т.е. конкретизации имеющихся фактов, данных и знаний по направлениям науки и техники. Признание информации как ресурса и появление понятия "информационный ресурс" дало толчок развитию нового научного направления – *информатики*. Информатика, как область науки и техники, связана со сбором и переработкой больших объемов информации на основе современных программно-аппаратных средств вычислительной техники и техники связи. Информатика изучает свойства информационных ресурсов, разрабатывает эффективные методы и средства их организации, преобразования и применения. На основе достижений информатики формируются новые методы и алгоритмы преобразования информации, при которых неквалифицированный в области вычислительной техники пользователь на языке, близком к естественному, может общаться с вычислительной средой для решения требуемых практических задач. На пользовательском уровне информатика дает основу для создания современных информационных систем, таких как автоматизированные системы управления, автоматизированные системы научных исследований, информационно-справочные системы, интеллектуальные системы, системы управления реального времени и др. Учитывая, что техническими средствами информатики являются вычислительные средства, ее современное состояние и направления дальнейшего развития в значительной степени определяются перспективами создания, развития и внедрения персональных ЭВМ, сетей связи, языков общения пользователя с вычислительной техникой. Информатика, как область науки и техники, требует своего дальнейшего развития. В качестве основных направлений исследований в области информатики можно определить следующее: разработка новой информационной технологии проектирования систем; развитие интеллектуальных методов доступа пользователя к вычислительной среде; создание моделей анализа и синтеза информационных процессов; совершенствование программных и аппаратных средств вычислительной техники и техники связи; переход к интеллектуальным АСОИУ (автоматизированная система обработки информации управления) на основе гибридных экспертных систем.

Информационные меры, как правило, рассматриваются в трех аспектах: структурном, статистическом и семантическом.

В структурном аспекте рассматривается строение массивов информации и их измерение простым подсчетом информационных элементов или комбинаторным методом. Структурный подход применяется

для оценки возможностей информационных систем вне зависимости от условий их применения.

При статистическом подходе используется понятие энтропии как меры неопределенности, учитывающей вероятность появления и информативность того или иного сообщения. Статистический подход учитывает конкретные условия применения информационных систем.

Семантический подход позволяет выделить полезность или ценность информационного сообщения.

1.2 Структурная мера информации

Информация всегда представляется в виде сообщения. Элементарная единица сообщений — *символ*. Символы, собранные в группы, — *слова*. Сообщение, оформленное в виде слов или отдельных символов, всегда передается в материально-энергетической форме (электрический, световой, звуковой сигнал и т. д.).

Различают информацию *непрерывную* и *дискретную*.

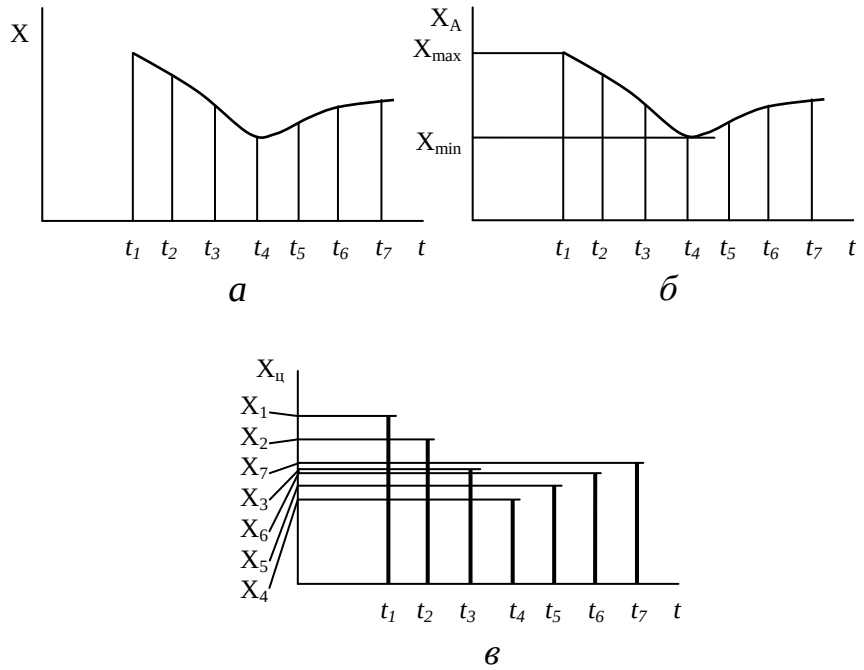


Рисунок 1.2 – Способы представления информации

Функция $x(t)$, изображенная на рисунке 1.2*а*, может быть представлена в непрерывном (рисунок 1.2*б*) и дискретном (рисунок 1.2*в*) видах. В непрерывном виде эта функция может принимать любые вещественные значения в данном диапазоне изменения аргумента t, т. е. множество значений непрерывной функции бесконечно. В дискретном виде функция $x(t)$ может принимать вещественные значения только при определенных значениях аргумента. Какой бы малый интервал дискретности (т.е. расстояние между соседними значениями аргумента) не выбирался,

множество значений дискретной функции для заданного диапазона изменений аргумента (если он не бесконечный) будет конечно (ограничено).

При использовании структурных мер информации учитывается только дискретное строение сообщения, количество содержащихся в нем информационных элементов, связей между ними. При структурном подходе различаются *геометрическая, комбинаторная* и *аддитивная* меры информации.

Геометрическая мера предполагает измерение параметра геометрической модели информационного сообщения (длины, площади, объема и т.п.) в дискретных единицах. Например, геометрической моделью информации может быть линия единичной длины (рисунок 1.3*а* – одноразрядное слово, принимающее значение 0 или 1), квадрат (рисунок 1.3*б* — двухразрядное слово) или куб (рис 1.3*в* — трехразрядное слово). Максимально возможное количество информации в заданных структурах определяет информационную емкость модели (системы), которая определяется как сумма дискретных значений по всем измерениям (координатам).

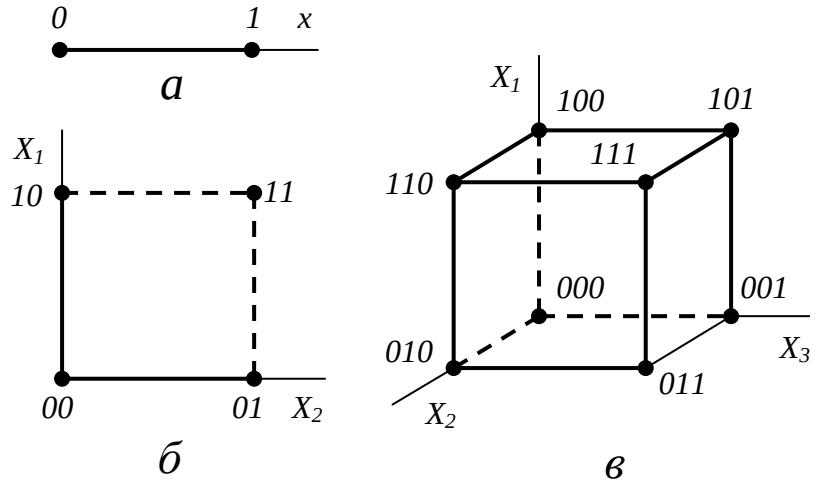


Рисунок 1.3 – Геометрическая модель информации

В комбинаторной мере количество информации определяется как число комбинаций элементов (символов). Возможное количество информации совпадает с числом возможных сочетаний, перестановок и размещений элементов. Комбинирование символов в словах, состоящих только из 0 и 1, меняет значения слов. Рассмотрим две пары слов 100110 и 001101, 011101 и 111010. В них проведена перестановка крайних разрядов (изменено местоположение знакового разряда в числе — перенесен слева направо).

Аддитивная мера (*мера Хартли*), в соответствии с которой количество информации измеряется в двоичных единицах — битах (наиболее распространена). Вводятся понятия глубины *q* и длины *n* числа.

Глубина q числа – количество символов (*элементов*), *принятых для представления информации*. В каждый момент времени реализуется только один какой-либо символ.

Длина n числа – количество позиций, необходимых и достаточных для представления чисел заданной величины.

Далее понятие глубины числа будет трансформировано в понятие основания системы счисления. При заданных глубине и длине числа количество чисел, которое можно представить, $N = q^n$. Величина N неудобна для оценки информационной емкости. Введем логарифмическую меру, позволяющую, вычислять количество информации — *бит*:

$$I(g)=\log_2 N = n \log_2 q \qquad (1.1)$$

Следовательно, 1 бит информации соответствует одному элементарному событию, которое может произойти или не произойти. Такая мера количества информации удобна тем, что она обеспечивает возможность оперировать мерой как числом. Количество информации при этом эквивалентно количеству двоичных символов 0 или 1. При наличии нескольких источников информации общее количество информации

$$I(q_1, q_2, \ldots, q_k) = I(q_1) + I(q_2) + \ldots + I(q_k), \qquad (1.2)$$

где $I(q_k)$ – количество информации от источника k.

Логарифмическая мера информации позволяет измерять количество информации и используется на практике.

1.3 Статистическая мера информации

В статистической теории информации вводится более общая мера количества информации, в соответствии с которой рассматривается не само событие, а информация о нем. Этот вопрос глубоко проработан *К. Шенноном* в работе «Избранные труды по теории информации». Если появляется сообщение о часто встречающемся событии, вероятность появления которого близка к единице, то такое сообщение для получателя малоинформативно. Столь же малоинформативны сообщения о событиях, вероятность появления которых близка к нулю.

События можно рассматривать как возможные исходы некоторого опыта, причем все исходы этого опыта составляют *ансамбль*, или *полную группу* событий. К. Шеннон ввел понятие неопределенности ситуации, возникающей в процессе опыта, назвав ее *энтропией*. Энтропия ансамбля есть количественная мера его неопределенности и, следовательно, информативности, количественно выражаемая как средняя функция множества вероятностей каждого из возможных исходов опыта.

Пусть имеется N возможных исходов опыта, из них k разных типов, а i-й исход повторяется n_i раз и вносит информацию, количество которой оценивается как I_i,. Тогда средняя информация, доставляемая одним опытом,

$$I_{cp} = (n_1 I_1 + n_2 I_2 + \ldots + n_k I_k)/N \qquad (1.3)$$

Но количество информации в каждом исходе связано с его вероятностью p_i и выражается в двоичных единицах (битах) как $I_i=\log_2 (1/p_i)= = -\log_2 p_i$. Тогда

$$I_{ср} = [n_1(-\log_2 p_1) + \ldots + n_k(-\log_2 p_k)]/N \qquad (1.4)$$

Выражение (1.4) можно записать также в виде

$$I_{ср} = \frac{n_1}{N}(-\log_2 p_1) + \frac{n_2}{N}(-\log_2 p_2) + \ldots + \frac{n_k}{N}(-\log_2 p_1) \qquad (1.5)$$

Отношения n/N представляют собой частоты повторения исходов, а следовательно, могут быть заменены их вероятностями $n_i/N=p_i$, поэтому их средняя информация в битах

$$I_{ср} = p_1(-\log_2 p_1) + \ldots + p_k(-\log_2 p_k)],$$

или

$$I_{ср}= -\sum_{i=1}^{k} p_i \log_2 p_i = H\,. \qquad (1.6)$$

Полученную величину называют энтропией и обозначают обычно буквой H. Энтропия обладает следующими свойствами:

1. Энтропия всегда неотрицательна, так как значения вероятностей выражаются величинами, не превосходящими единицу, а их логарифмы – отрицательными числами или нулем, так что члены суммы (1.6) – неотрицательны.

2. Энтропия равна нулю в том крайнем случае, когда одно из p_i равно единице, а все остальные — нулю. Это тот случай, когда об опыте или величине все известно заранее и результат не дает новую информацию.

3. Энтропия имеет наибольшее значение, когда все вероятности равны между собой: $p_1 = p_2 = ... = p_k = 1/k$. При этом

$$H = -\log_2 (1/k) = \log_2 k\,.$$

4. Энтропия объекта AB, состояния которого образуются совместной реализацией состояний A и B, равна сумме энтропии исходных объектов A и B, т. е. $H(AB) = H(A) + H(B)$.

Если все события равновероятны и статистически независимы, то оценки количества информации, по Хартли и Шеннону, совпадают. Это свидетельствует о полном использовании информационной емкости системы. В случае неравных вероятностей количество информации, по Шеннону, меньше информационной емкости системы. Максимальное значение энтропии достигается при $p = 0{,}5$, когда два состояния равновероятны. При вероятностях $p = 0$ или $p = 1$, что соответствует полной невозможности или полной достоверности события, энтропия равна нулю.

Количество информации только тогда равно энтропии, когда неопределенность ситуации снимается полностью. В общем случае нужно считать, что количество информации есть уменьшение энтропии вследствие опыта или какого-либо другого акта познания. Если неопределенность снимается полностью, то информация равна энтропии: $I = H$.

В случае неполного разрешения имеет место частичная информация, являющаяся разностью между начальной и конечной энтропией: $I = H_1–H_2$.

Наибольшее количество информации получается тогда, когда полностью снимается неопределенность, причем эта неопределенность была наибольшей – вероятности всех событий были одинаковы. Это соответствует максимально возможному количеству информации I^1, оцениваемому мерой Хартли:

$I^1 = \log_2 N = \log_2(1/p) = -\log_2 p$,

где N — число событий; p — вероятность их реализации в условиях равной вероятности событий.

Таким образом, $I^1 = H_{max}$.

Абсолютная избыточность информации $D_{абс}$ представляет собой разность между максимально возможным количеством информации и энтропией: $D_{абс} = I^1 - H$, или $D_{абс} = H_{max} - H$.

Пользуются также понятием относительной избыточности

$$D = (H_{max} - H)/H_{max} \qquad (1.7)$$

1.4 Семантическая мера информации

Вычислительные машины обрабатывают и преобразуют информацию разного содержания — от числовых данных до сочинения музыки и стихов. Вся эта информация изображается соответствующими символами. Оценка содержания разнохарактерной информации — весьма сложная проблема.

Среди семантических мер наиболее распространены содержательность, логическое количество, целесообразность и существенность информации.

Содержательность события выражается через функцию меры $m(i)$ – содержательности его отрицания. Оценка содержательности основана на математической логике, в которой логические функции истинности $m(i)$ и ложности $m(\bar{i})$ имеют формальное сходство с функциями вероятностей события $p(i)$ и антисобытия $q(i)$ в теории вероятностей.

Как и вероятность, содержательность события изменяется в пределах $0 \le m(i) \le 1$.

Логическое количество информации I_{nf} , сходное со статистическим количеством информации, вычисляется по выражению:

$$I_{nf} = \log_2 [1/m(i)] = -\log_2 m(\bar{i}) .$$

Отличие статистической оценки от логической состоит в том, что в первом случае учитываются вероятности реализации тех или иных событий, что приближает к оценке смысла информации.

Если информация используется в системах управления, то ее полезность целесообразно оценивать по тому эффекту, который она оказывает на результат управления.

Мера целесообразности информации определяется как изменение вероятности достижения цели при получении дополнительной информации. Полученная информация может быть пустой, т. е. не изменять вероятности достижения цели, и в этом случае ее мера равна нулю. В других случаях полученная информация может изменять положение дела в худшую сторону,

т.е. уменьшить вероятность достижения цели, и тогда она будет дезинформацией, измеряющейся отрицательным значением количества информации. Наконец, в благоприятном случае получается добротная информация, которая увеличивает вероятность достижения цели и измеряется положительной величиной количества информации.

Мера целесообразности в общем виде может быть аналитически выражена в виде соотношения

$$I_{\text{цел}} = \log_2 p_1 - \log_2 p_0 = \log_2 \frac{p_1}{p_0}, \qquad (1.8)$$

где p_0 и p_1 – начальная (до получения информации) и конечная (после получения информации) вероятности достижения цели.

Следует различать существенность самого события; существенность времени совершения события или его наблюдения (рано—поздно—момент); существенность координаты совершения события.

Измерение некоторого параметра X можно характеризовать несколькими функциями величины x: вероятностью $p(x)$, погрешностью измерения $\varepsilon(x)$ и существенностью $c(x)$. Каждой из этих функций можно поставить в соответствие определенную меру информации. Мерой Хартли оценивается функция погрешности ε при фиксированных значениях функции вероятности (p = const) и существенности (c = const). Мерой Шеннона оценивается функция вероятности (p = var) при фиксированных значениях функций погрешности (ε = const) и существенности (c = const) . Мера существенности относится к ситуации с фиксированными функциями погрешности (ε = const) и вероятности (p = const) . Можно ввести функции существенности: c_x, зависящие от x; c_T, c_N, зависящие от времени T и пространства (канала) N.

1.5 Преобразование информации

Информационное сообщение всегда связано с источником информации, приемником информации и каналом передачи.

Дискретные сообщения состоят из конечного множества элементов, создаваемых источником последовательно во времени. Набор элементов (символов) составляет алфавит источника.

Непрерывные сообщения задаются какой-либо физической величиной, изменяющейся во времени. Получение конечного множества сообщений за конечный промежуток времени достигается путем дискретизации (во времени), *квантования* (по уровню) (см. рисунок 1.2).

В большинстве случаев информация о протекании того или иного физического процесса вырабатывается соответствующими датчиками в виде сигналов, непрерывно изменяющихся во времени. Переход от аналогового представления сигнала к цифровому дает в ряде случаев значительные преимущества при передаче, хранении и обработке информации. Преобразование осуществляется с помощью специальных устройств –

преобразователей непрерывных сигналов и может быть выполнено дискретизацией во времени и квантованием по уровню.

Рассмотрим разновидности сигналов, которые описываются функцией $x(t)$.

1. Непрерывная функция непрерывного аргумента. Значения, которые могут принимать функция $x(t)$ и аргумент t, заполняют промежутки $(x_{\min}, x_{\max})$ и $(-T, T)$ соответственно.

2. Непрерывная функция дискретного аргумента. Значения функции $x(t)$ определяются лишь на дискретном множестве значений аргумента t_i, $i=0\pm1\pm2, \ldots$ Величина $x(t_i)$ может принимать любое значение в интервале $(x_{\min}, x_{\max})$.

3. Дискретная функция непрерывного аргумента. Значения, которые может принимать функция $x(t)$, образуют дискретный ряд чисел $x_1, x_2, \ldots, x_k$. Значение аргумента t может быть любым в интервале $(-T, T)$.

4. Дискретная функция дискретного аргумента. Значения, которые могут принимать функция $x(t)$ и аргумент t, образуют дискретные ряды чисел $x_1, x_2, \ldots, x_k$ и $t_1, t_2, \ldots, t_k$, заполняющие интервалы $(x_{\min}, x_{\max})$ и $(-T,T)$ соответственно.

Первая из рассмотренных разновидностей принадлежит непрерывным сигналам, вторая и третья — дискретно-непрерывным, а четвертая — дискретным сигналам.

Операцию, переводящую информацию непрерывного вида в информацию дискретного вида, называют *квантованием по времени*, или *дискретизацией*. Следовательно, дискретизация состоит в преобразовании сигнала $x(t)$ непрерывного аргумента t в сигнал $x(t_i)$ дискретного аргумента t_i.

Квантование по уровню состоит в преобразовании непрерывного множества значений сигнала $x(t_i)$ в дискретное множество значений x_k, $k = 0,1,\ldots, (m - 1)$; $x_k \in (x_{\min}, x_{\max})$ (третий вид сигнала).

Совместное применение операций дискретизации и квантования по уровню позволяет преобразовать непрерывный сигнал $x(t)$ в дискретный по координатам x и t (четвертая разновидность).

В результате дискретизации исходная функция $x(t)$ заменяется совокупностью отдельных значений $x(t_i)$. По значениям функции $x(t_i)$ можно восстановить исходную функцию $x(t)$ с некоторой погрешностью. Функцию, полученную в результате восстановления (интерполяции) по значениям $x(t_i)$, будем называть *воспроизводящей* и обозначать $V(t)$.

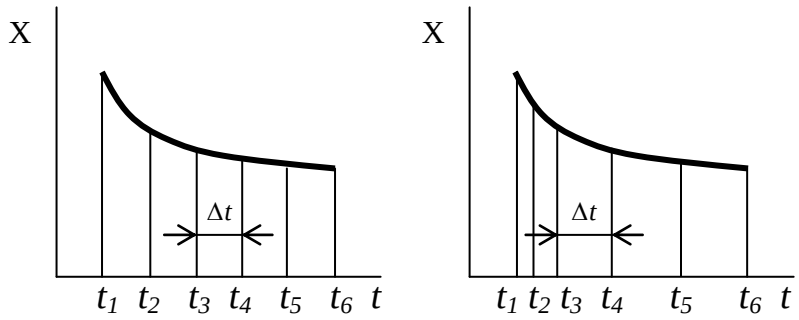
При дискретизации сигналов приходится решать вопрос о том, как часто следует проводить *отсчеты* функции, т. е. каков должен быть шаг дискретизации $\Delta t_i = t_i - t_{i-1}$. При малых шагах дискретизации количество отсчетов функции на отрезке обработки будет большим и точность воспроизведения — высокой. При больших шагах дискретизации количество отсчетов уменьшается, но при этом, как правило, снижается точность восстановления. Оптимальной является такая дискретизация, которая обеспечивает представление исходного сигнала с заданной точностью при минимальном количестве отсчетов.

Методы дискретизации и восстановления информации классифицируются в зависимости от регулярности отсчета, критерия оценки точности дискретизации и восстановления, вида базисной функции, принципа приближения.

Регулярность отсчета определяется равномерностью дискретизации.

Дискретизация называется равномерной (рисунок 1.4*а*), если длительность интервалов Δt_i = const на всем отрезке обработки сигнала. Методы равномерной дискретизации широко применяют, так как алгоритмы и аппаратура для их реализации достаточно просты. Однако при этом возможна значительная избыточность отсчетов.

Дискретизация называется неравномерной (рисунок 1.4*б*), если длительность интервалов между отсчетами Δt_i , различна, т. е. Δt_i = var . Выделяют две группы неравномерных методов: адаптивные и программируемые. При адаптивных методах интервалы Δt_i , изменяются в зависимости от текущего изменения параметров сигналов. При программируемых методах интервалы Δt_i , изменяются либо оператором на основе анализа поступающей информации, либо в соответствии с заранее установленной программой работы.



а *б*

Рисунок 1.4 – Способы дискретизации информации

Критерии оценки точности дискретизации сигнала выбираются получателем информации и зависят от целевого использования сигнала и возможностей аппаратной (программной) реализации. Чаще других используются критерий наибольшего отклонения, среднеквадратический, интегральный и вероятностный.

Тип базисных (приближающих, воспроизводящих) функций в основном определяется требованиями ограничения сложности устройств (программ) дискретизации и восстановления сигналов.

Воспроизводящие функции $v(t)$ обычно совпадают с приближающими функциями $p(t)$, хотя в общем случае они могут отличаться друг от друга. Чаще всего для дискретизации и восстановления используют ряды Фурье и Котельникова, полиномы Чебышева и Лежандра, степенные полиномы, функции Уолша и Хаара, гипергеометрические функции.

При равномерной дискретизации шаг Δt и частота отсчетов F_0 — постоянные величины (рисунок 1.4*а*). Модель равномерной дискретизации очень хорошо подходит к модели синхронных автоматов. *Теорема Котельникова* позволяет осуществлять выбор шага дискретизации, что существенным образом может повлиять на количество и скорость поступления информации для обработки.

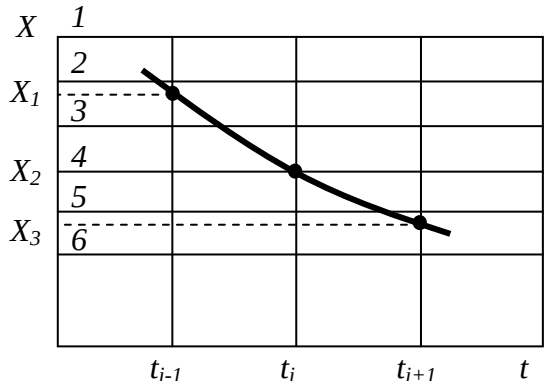


Рисунок 1.5 – Квантование по уровню

Квантование по уровню состоит в преобразовании непрерывных значений сигнала $x(t_i)$ в моменты отсчета t_i, в дискретные значения (рисунок 1.5). В соответствии с графиком изменения функции $x(t)$ ее истинные значения представляются в виде заранее заданных дискретных уровней *1*, *2*, *3*, *4*, *5* или *6*. Функция в моменты отсчета может задаваться или точно (значение x_2 – уровень *4*), или с некоторой погрешностью (значение x_1 — уровень *2*, значение x_3 — уровень *6*).

Квантование по уровню может быть равномерным и неравномерным в зависимости от величины шага квантования. Под шагом (интервалом) квантования δ_m понимается разность $\delta_m = x_m - x_{m-1}$, где x_m, x_{m-1} – соседние уровни квантования.

Уровень квантования для заданного значения сигнала $x(t)$ можно выразить двумя способами:

1) сигнал $x(t_i)$ отождествляется с ближайшим уровнем квантования;
2) сигнал $x(t_i)$ отождествляется с ближайшим меньшим (или большим) уровнем квантования.

Так как в процессе преобразования значение сигнала $x(t)$ отображается уровнем квантования x_m , а каждому уровню m может быть поставлен в соответствие свой номер (число), то при передаче или хранении информации можно вместо истинного значения величины x_m использовать соответствующее число m. Истинное значение уровня квантования легко восстановить, зная масштаб по шкале x. Для представления m уровней квантования с помощью неизбыточного равномерного кода потребуется $n=\log_2 m$ разрядов. Такое преобразование сопровождается шумами или погрешностью квантования. Погрешность квантования связана с заменой истинного значения сигнала $x(t_i)$ значением, соответствующим уровню

квантования x_m . Максимальная погрешность квантования зависит от способа отождествления сигнала с уровнем квантования. Для первого из рассмотренных способов она равна $0{,}5\delta_m$, для второго – δ_m.

Чем меньше шаг квантования, тем меньше погрешность квантования. Можно принять, что погрешность квантования в пределах шага квантования имеет равновероятный закон распределения, т. е. любое значение функции в пределах шага будет равновероятным.

Наиболее часто используются степенные алгебраические полиномы вида $V(t)=\sum_{i=0}^{n} a_i t^i$, где n — степень полинома, a_i — действительные коэффициенты. Из этого класса функций наиболее полно исследовано применение полиномов нулевой и первой степени. Алгебраические полиномы удобны для программирования и обработки на ЭВМ.

Выбор оптимальной функции представляет определенные трудности, так как при решении задачи минимизации числа дискретных характеристик для описания сигнала с заданной точностью должны учитывать сложность аппаратуры (программ), допустимое время задержки в выдаче информации и другие факторы.

Метод дискретизации при преобразовании непрерывной информации в дискретную влияет на количество информации, которую надо хранить или преобразовывать в ЭВМ. Важна теорема Котельникова, согласно которой функция, имеющая ограниченный спектр частот, полностью определяется дискретным множеством своих значений, взятых с частотой отсчетов:

$$F_0 = 2f_m, \tag{1.9}$$

где $f_m = 2\pi\omega_m$ – максимальная частота в спектре частот $S(j\omega)$ сигнала $x(t)$; ω_m – угловая скорость.

Функция $x(t)$ воспроизводится без погрешностей по точным значениям $x(t_i)$ в виде ряда Котельникова:

$$x(t) = \sum_{k=-\infty}^{0} x(k_{\Delta t}) \frac{\sin \omega_m (t - k_{\Delta t})}{(t - k_{\Delta t})}, \tag{1.10}$$

где Δt – шаг дискредитации.

Теорема Котельникова справедлива для сигналов с ограниченным спектром. Реальные сигналы — носители информации — имеют конечную длительность. Спектр таких сигналов не ограничен, т. е. реальные сигналы не соответствуют в точности модели сигнала с ограниченным спектром, и применение теоремы Котельникова к реальным сигналам связано с погрешностями при восстановлении сигналов по формуле (1.10) и неопределенностью выбора шага дискретизации или частоты отсчетов.

Для практических задач, однако, идеально точное восстановление функций не требуется, необходимо лишь восстановление с заданной точностью. Поэтому теорему Котельникова можно рассматривать как приближенную для функций с неограниченным спектром. На практике частоту отсчетов часто определяют по формуле

$$F_0 = 2f_{max}k_3, \tag{1.11}$$

где k_3 — коэффициент запаса (обычно $1{,}5 < k_3 < 6$); f_{max} — максимальная допустимая частота в спектре сигнала $x(t)$, например, с учетом доли полной энергии, сосредоточенной в ограниченном частотой спектре сигнала.

Из вышеизложенного следует, что преобразование непрерывной информации в дискретную может сопровождаться сжатием информации (уменьшением ее количества). Квантование по уровню — один из способов сжатия информации.

Квантование и дискретизация находят широкое применение в преобразователях информации, используемых при связи ЭВМ с конкретными объектами (процессами).

1.6 Формы представления информации

Информация всегда представляется в виде сообщения, которое передается некоторой физической средой. Носителем информации может быть любая предметная среда, которая меняет состояние в зависимости от передаваемой информации. Это может быть бумага, на которой информация изображается либо знаками, либо специальными отметками (например, перфорация); магнитный материал (лента, диск и т. п.), состояние которого меняется с помощью магнитной головки; электрический сигнал, у которого изменяется какой-либо параметр (частота, амплитуда); оптический носитель, работа которого основана на оптическом методе считывания.

Различают две формы представления информации: *статическую* I_c, (рисунок 1.6а) и *динамическую* $I_д$ (рисунок 1.6*б*). Возможность передачи сообщения посредством электрического сигнала реализуется с помощью канала связи, соединяющего источник и приемник информации (рисунок 1.7). Чтобы передать информацию, необходимо ее предварительно преобразовать.

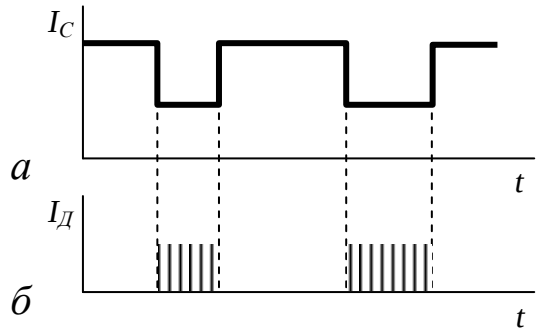


Рисунок 1.6 – Формы представления информации

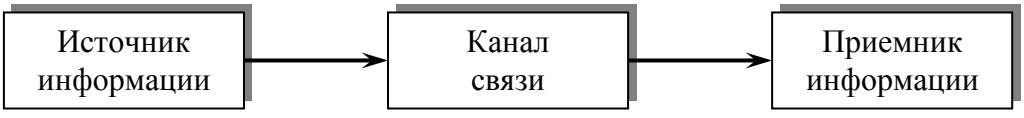


Рисунок 1.7 – Информационная модель канала связи

Кодирование — преобразование сообщения в форму, удобную для передачи по данному каналу. В качестве простого примера можно привести передачу сообщения в виде телеграммы. Все символы кодируются с помощью телеграфного кода.

Декодирование — операция восстановления принятого сообщения. В систему связи необходимо ввести устройства для кодирования и декодирования информации (рисунок 1.8). Теоретическое обоснование таких систем дал в своих работах К. Шеннон. Рядом теорем он показал эффективность введения кодирующих и декодирующих устройств, назначение которых состоит в согласовании свойств источника информации со свойствами канала связи. Одно из них (кодирующее устройство, или кодер) должно обеспечить такое кодирование, при котором путем устранения избыточности информации существенно снижается среднее число символов, приходящееся на единицу сообщения. При отсутствии помех это непосредственно дает выигрыш во времени передачи или в объеме запоминающего устройства. Такое кодирование называют *эффективным* (или *оптимальным*), так как оно повышает эффективность системы. При наличии помех в канале передачи оно позволяет преобразовать входную информацию в последовательность символов, наилучшим образом отвечающую задачам дальнейшего преобразования. Другое кодирующее устройство (кодер канала) обеспечивает заданную достоверность при передаче или хранении информации путем введения дополнительно избыточности информации. Такое кодирование называют *избыточным* или *помехоустойчивым*. Помехоустойчивость достигается учетом не только интенсивности помехи, но и ее статистических закономерностей.

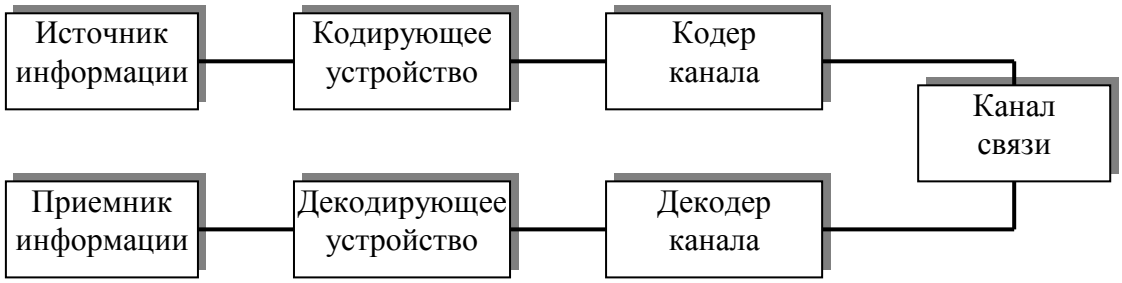


Рисунок 1.8 – Информационная модель канала связи с шумами

Целесообразность устранения избыточности сообщения методами эффективного кодирования с последующим перекодированием помехоустойчивым кодом обусловлена тем, что избыточность источника сообщения в большинстве случаев не согласована со статистическими закономерностями помехи в канале связи и поэтому не может быть полностью использована для повышения достоверности принимаемого сообщения. Кроме того, избыточность источника сообщений иногда является следствием ряда причин.

Если избыточность источника сообщений мала и помехи в канале связи практически отсутствуют, то введение как кодера источника, так и кодера канала нецелесообразно. Если избыточность источника сообщений высока, а помехи весьма малы, то целесообразно ввести кодер источника. Если избыточность источника мала, а помехи велики, то целесообразно ввести кодер канала. При большой избыточности и высоком уровне помех целесообразно ввести оба дополнительных кодирующих (и декодирующих) устройства. Большинство кодов, используемых при кодировании информации без учета статистических свойств источника и помехи в канале связи, основано на системах счисления.

1.7 Передача информации

Современные вычислительные средства часто используются в составе вычислительных систем или сетей. В этих случаях необходимо решать вопросы не только эффективного представления информации, но также вопросы передачи информации по каналам связи без искажений. В качестве каналов связи (рисунок1.9) могут использоваться: непосредственная связь (НС) пользователя вычислительными средствами, телефонный канал (ТлК), телеграфный канал (ТгК), радиоканал (РК), телевизионный канал (ТвК), другие виды связи.

Вид канала определяет характер и величину помех, которые при этом появляются. Поэтому инженер-проектировщик ЭВМ должен учитывать эти обстоятельства при разработке технических, математических и программных средств.

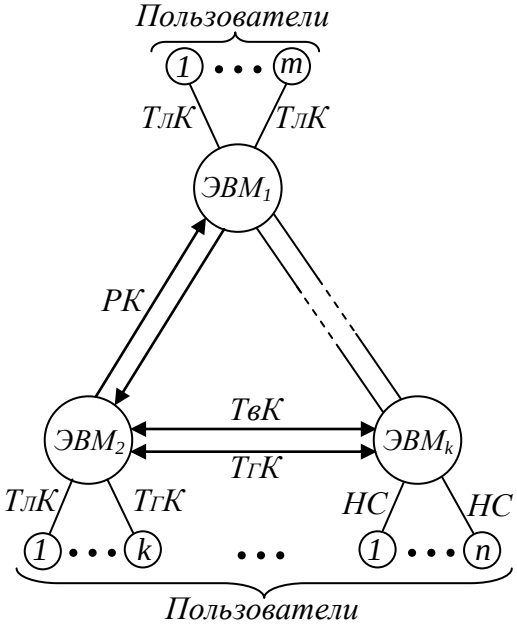


Рисунок 1.9 – Каналы связи в вычислительных сетях

Рассмотрим некоторые общие вопросы, возникающие при передаче информации.

Передача информации по каналу без помех

Если через канал связи без помех передается последовательность дискретных сообщений длительностью T, то скорость передачи информации по каналу связи (бит/с)

$$\nu = \lim_{T\to\infty}(I/T) \qquad (1.12)$$

где I – количество информации, содержащейся в последовательности сообщений.

Предельное значение скорости передачи информации называется пропускной способностью канала связи без помех $c = \nu_{max}$.

Количество информации в сообщениях максимально при равной вероятности состояний. Тогда

$$\nu = \lim_{T\to\infty}\log_2 k/T. \qquad (1.12)$$

Скорость передачи информации в общем случае зависит от статистических свойств сообщений и параметров канала связи.

Пропускная способность – характеристика канала связи, которая не зависит от скорости передачи информации. Количественно пропускная способность канала связи выражается максимальным количеством двоичных единиц информации, которое данный канал связи может передать за одну секунду.

Для наиболее эффективного использования канала связи необходимо, чтобы скорость передачи информации была как можно ближе к пропускной способности канала связи. Если скорость поступления информации на вход канала связи превышает пропускную способность канала, то по каналу будет передана не вся информация. Основное условие согласования источника информации и канала связи

$$v \le c. \qquad (1.14)$$

Согласование осуществляется путем соответствующего кодирования сообщений. Доказано, что если скорость информации, вырабатываемой источником сообщений $\nu_и$, достаточно близка к пропускной способности канала c, т. е. $\nu_и = c - \varepsilon$, где ε — сколь угодно малая величина, то всегда можно найти такой способ кодирования, который обеспечит передачу сообщений, вырабатываемых источником, причем скорость передачи информации будет весьма близка к пропускной способности канала.

Верно и обратное утверждение: невозможно обеспечить длительную передачу всех сообщений, если поток информации, вырабатываемый источником, превышает пропускную способность канала.

Если ко входу канала подключен источник сообщений с энтропией, равной пропускной способности канала связи, то считается, что источник согласован с каналом. Если энтропия источника меньше пропускной способности канала, что может быть в случае неравновероятности состояний

источника, то источник не согласован с каналом, т. е. канал используется не полностью.

Согласование в статистическом смысле достигается с помощью статистического кодирования. Оно позволяет повысить энтропию передаваемых сообщений до величины, которая получается, если символы новой последовательности равновероятны. При этом число символов в последовательности будет сокращено. В результате источник информации согласуется с каналом связи.

Передача информации по каналу с помехами

При передаче информации через канал с помехами сообщения искажаются, и на приемной стороне нет уверенности в том, что принято именно то сообщение, которое передавалось. Следовательно, сообщение недостоверно, вероятность правильности его после приема не равна единице. В этом случае количество получаемой информации уменьшается на величину неопределенности, вносимой помехами, т. е. вычисляется как разность энтропии сообщения до и после приема: $I' = H(i) - H_i(i)$, где $H(i)$ – энтропия источника сообщений; $H_i(i)$ – энтропия сообщений на приемной стороне.

Таким образом, скорость передачи по каналу связи с помехами

$$v' = \lim_{T \to \infty} \frac{H(i) - H_i(i)}{T}. \qquad (1.15)$$

Пропускной способностью канала с шумами называется максимальная скорость передачи информации при условии, что канал связи без помех согласован с источником информации:

$$c = \lim_{T \to \infty} \frac{I_{\max}}{T}.$$

Если энтропия источника информации не превышает пропускной способности канала ($H \le c$), то существует код, обеспечивающий передачу информации через канал с помехами со сколь угодно малой частотой ошибок или сколь угодно малой недостоверностью. Пропускная способность канала связи при ограниченной средней мощности аналогового сигнала

$$c = F_m \log_2 (1 + W_c / W_m), \qquad (1.16)$$

где F_m – полоса частот канала (Гц); W_c – средняя мощность сигнала; W_m — средняя мощность помех (равномерный спектр) с нормальным законом распределения амплитуд в полосе частот канала связи.

Следовательно, можно передавать информацию по каналу с помехами без ошибок, если скорость передачи информации меньше пропускной способности канала, определяемой формулой (1.16). Для скорости $v > c$ при любой системе кодирования частота ошибок принимает конечное значение, причем оно растет с увеличением значения v. Из выражения (1.16) следует, что для канала с весьма высоким уровнем шумов ($W_m >> W_c$) максимальная скорость передачи близка к нулю.

1.8 Общая характеристика фаз преобразования информации

Основой функционирования ИС является информационный процесс, характеризующийся определенными фазами преобразования информации.

Среди фаз обращения информации, следует отметить, прежде всего, **фазу подготовки** информации. Подготовка информации может осуществляться вручную, машинным способом, с использованием различных видов носителей. Для увеличения быстродействия целесообразно осуществлять этап подготовки, используя либо машинные носители информации, либо такие сигналы, которые способны передаваться непосредственно от источника в канал связи. На этапе подготовки информация, снимаемая с объекта управления или подготовленная в результате действий оператора, наносится на некоторый носитель по определенному правилу и далее включается в информационный процесс. Другой фазой является **регистрация** информации, осуществляемая с целью образования документа, в котором информация дана в формализованном виде. Этот документ может храниться и использоваться при последующем управлении.

Следующими фазами преобразования являются **сбор и передача** информации. Сбор информации обычно осуществляется с территориально разнесенных точек объекта управления, например в пределах цеха, предприятия, отрасли промышленности, а также и с более отдаленных объектов. Сбор информации нередко включает в себя и неразрывно связан с передачей информации, которая состоит в переносе информации на значительные расстояния посредством дополнительного преобразования исходных сообществ в сигналы, способные по своим физическим свойствам передаваться по выбранным каналам связи. При передаче информации формируются дополнительные сигналы, а поэтому на приемной стороне должны использоваться специальные средства, осуществляющие выявление сигналов по определенным алгоритмам.

Информация, передаваемая по каналам, в дальнейшем используется при принятии решения, поэтому она должна быть обработана. **Обработка** информации в ИС производится с помощью ЭВМ, где централизуются функции обработки, и на основе отдельных моделей ситуации человеком с помощью детерминированных или неформализованных способов осуществляется принятие решения. При этом информации подвергаются цифровым и аналоговым преобразованиям. В процессе обработки возможны промежуточные этапы хранения информации с использованием оперативных и долговременных запоминающих устройств, построенных на различных технических средствах. На этапе хранения информации возникает весьма серьезная задача систематизации имеющейся информации. Из информации формируется набор данных, создается банк данных. В итоге возникают проблемы создания информационных массивов, а также поиска и организации информационных массивов таким образом, чтобы обеспечить заданное быстродействие при записи и выводе информации из массива.

Поскольку выработка управляющих воздействий немыслима без дальнейшего использования информации в контуре управления, то важной задачей является **вывод** информации в соответствующем виде и ее **воспроизведение.** Воспроизведение информации требуется прежде всего тогда, когда в информационном процессе активное участие принимает человек и ему необходимо получить качественные и количественные характеристики выходной информации в наглядной форме. Для этого используются специальные технические средства цифрового и графического характера, а также экраны с различного типа мнемосхемами, индикаторами и т. д., которые способны воздействовать на органы чувств человека. Кроме того, информация может быть выдана в виде документов, которые далее используются в ИС организационно-экономического характера. Отдельные данные из управляющей вычислительной машины с помощью цифроаналоговых преобразователей могут выдаваться непосредственно в виде электрических сигналов, которые осуществляют либо регулирующие, либо управляющие, либо защитные действия и тем самым корректируют технологический процесс.

Наряду с крупными этапами или фазами преобразования информации существуют более мелкие операции, связанные с отдельными воздействиями на информацию для получения каких-то данных по заранее известным алгоритмам. Сюда, прежде всего, можно отнести такую дополнительную операцию как **классификация.** Классификация оказывается необходимой в ряде случаев, например при хранении информации, когда данные, накапливаемые случайным образом, должны храниться в форме, удобной для последующего их извлечения.

При этом выбираются определенные классификационные признаки, которые вносятся в саму информацию и хранятся вместе с основной информацией. Весьма существенной операцией является **синтез.** Данная операция необходима в случае, когда требуется объединить отдельные составляющие данные по одному и тому же вопросу в совокупность данных для получения единого логически связанного слова.

Независимо от фазы преобразования информации каждый вид ее обладает определенными характеристиками, среди которых полезно выделить связанные с функционированием ИС следующие характеристики.

Цель информации. Под целью информации можно понимать назначение процесса информирования, выработку и применение решения, выдачу команд и т. д.

Способ передачи и *формат информации.* Следует отметить, что формат информации бывает различным в смысле приемлемости по отношению к человеку и машине. Целесообразно при этом, чтобы человек получал большую часть информации в виде документа, где записан и систематизирован материал, или на экране монитора.

Избыточность информации. При подготовке информации возникает определенная избыточность, связанная с периодичностью ввода информации, особенно при вводе отдельных точек непрерывной величины. Избыточность

может иметь и источник дискретной информации, когда отдельные выдаваемые сообщения взаимно зависимы. Наличие исходной избыточности уменьшает быстродействие системы, увеличивает форматы сообщений, и поэтому одна из задач, которую приходится решать - это устранение первичной избыточности информации. Вместе с тем в фазе передачи информации избыточность является средством, полезным для борьбы с внешними возмущающими воздействиями (помехами), и при правильном выборе уровня вводимой избыточности и алгоритма построения сигнала в канале связи получаем повышение достоверности передачи информации по каналу.

Время преобразования информации является одной из основных характеристик функционирования комплекса технических средств. Оно зависит как от формата информации, который определяется свойствами технических средств, так и от алгоритма управления информационными потоками в информационной сети. В целом возникают временные задержки, которые вызывают старение информации и снижение ее ценности, если информация используется в системе оперативного управления производством. В связи с этим быстродействие есть одна из важнейших характеристик подсистем комплекса технических средств ИС.

Периодичность появления информации зависит непосредственно от конкретной функциональной подсистемы, по которой вводится информация. Периодичность может изменяться, при этом существует информация, которая должна обрабатываться в реальном масштабе времени, а также информация, которая может обрабатываться спустя значительный отрезок времени после ее возникновения, что особенно характерно для подсистем перспективного планирования, технической подготовки производства и др.

Верность информации - одна из основных характеристик, информации на любой фазе ее преобразования. В зависимости от верности информации определяется степень доверия к информации в процессе принятия решения, и верностью исходных данных определяется эффективность функционирования ИС. Поэтому подготовка информации, точно отражающей конкретный производственный процесс, является одним из актуальных вопросов при проектировании самой системы. Вместе с тем построение комплекса технических средств ИС, а также математического и информационного обеспечения в значительной степени осуществляется с учетом требований верности информации при функционировании ИС. Таким образом, наличие многих фаз преобразования информации и отдельных операций по преобразованию на каждой фазе вызывает появление различных форм представления информации в машине и для человека-оператора, но во всех случаям информация должна поставляться верно и своевременно.

Контрольные вопросы

1. Какими свойствами обладает информация?
2. Какие аспекты информации вы знаете?
3. Перечислите известные вам виды информации.
4. Перечислите известные вам меры информации.
5. Какие способы дискретизации информации вы знаете?
6. Что такое кодирование и декодирование информации?
7. Как происходит передача информации по каналу без помех?
8. Как происходит передача информации по каналу с помехами?
9. Перечислите фазы преобразования информации.

2. АЛГОРИТМИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

Слово «Алгоритм» происходит от algorithmi - латинского написания имени аль-Хорезми, под которым в средневековой Европе знали величайшего математика из Хорезма (город в современном Узбекистане) Мухаммеда бен Мусу, жившего в 783-850 гг. В своей книге «Об индийском счете» он сформулировал правила записи натуральных чисел с помощью арабских цифр и правила действий над ними столбиком. В дальнейшем *алгоритмом* стали называть точное предписание, определяющее последовательность действий, обеспечивающую получение требуемого результата из исходных данных. Алгоритм может быть предназначен для выполнения его человеком или автоматическим устройством. Создание алгоритма, пусть даже самого простого, - процесс творческий. Он доступен исключительно живым существам, а долгое время считалось, что только человеку. Другое дело - реализация уже имеющегося алгоритма. Ее можно поручить субъекту или объекту, который не обязан вникать в существо дела, а возможно, и не способен его понять. Такой субъект или объект принято называть *формальным исполнителем*. В роли формального исполнителя может выступать и человек, но в первую очередь – различные автоматические устройства, и компьютер в том числе. Каждый алгоритм создается в расчете на вполне конкретного исполнителя. Те действия, которые может совершать исполнитель, называются его *допустимыми действиями*. Совокупность допустимых действий образует *систему команд исполнителя*. Алгоритм должен содержать только те действия, которые допустимы для данного исполнителя.

2.1 Свойства алгоритмов

Данное выше определение алгоритма нельзя считать строгим - не вполне ясно, что такое «точное предписание» или «последовательность действий, обеспечивающая получение требуемого результата». Поэтому обычно формулируют несколько общих свойств алгоритмов, позволяющих отличать алгоритмы от других инструкций. Такими свойствами являются:

- *дискретность* (прерывность, раздельность) - алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов. Каждое действие, предусмотренное алгоритмом, исполняется только после того, как закончилось исполнение предыдущего;
- *определенность* - каждое правило алгоритма должно быть четким, однозначным и не оставлять места для произвола. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче;
- *результативность* (конечность) - алгоритм должен приводить к решению задачи за конечное число шагов;

- *массовость* - алгоритм решения задачи разрабатывается в общем виде, то есть, он должен быть применим для некоторого класса задач, различающихся только исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется *областью применимости* алгоритма.

Правила выполнения арифметических операций или геометрических построений представляют собой алгоритмы. При этом остается без ответа вопрос, чем же отличается понятие алгоритма от таких понятий, как «метод», «способ», «правило». Можно даже встретить утверждение, что слова «алгоритм», «способ», «правило» выражают одно и то же (т.е. являются синонимами), хотя такое утверждение, очевидно, противоречит «свойствам алгоритма».

Само выражение «свойства алгоритма» не совсем корректно. Свойствами обладают объективно существующие реальности. Можно говорить, например, о свойствах какого-либо вещества. Алгоритм – искусственная конструкция, которую мы сооружаем для достижения своих целей. Чтобы алгоритм выполнил свое предназначение, его необходимо строить по определенным правилам. Поэтому нужно говорить все же не о свойствах алгоритма, а о правилах построения алгоритма, или о требованиях, предъявляемых к алгоритму.

Первое правило – при построении алгоритма, прежде всего, необходимо задать множество объектов, с которыми будет работать алгоритм. Формализованное (закодированное) представление этих объектов носит название данных. Алгоритм приступает к работе с некоторым набором данных, которые называются входными, и в результате своей работы выдает данные, которые называются выходными. Таким образом, алгоритм преобразует входные данные в выходные.

Это правило позволяет сразу отделить алгоритмы от «методов» и «способов». Пока мы не имеем формализованных входных данных, мы не можем построить алгоритм.

Второе правило – для работы алгоритма требуется память. В памяти размещаются входные данные, с которыми алгоритм начинает работать, промежуточные данные и выходные данные, которые являются результатом работы алгоритма. Память является дискретной, т.е. состоящей из отдельных ячеек. Поименованная ячейка памяти носит название *переменной*. В теории алгоритмов размеры памяти не ограничиваются, т.е. считается, что мы можем предоставить алгоритму любой необходимый для работы объем памяти.

В школьной «теории алгоритмов» эти два правила не рассматриваются. В то же время практическая работа с алгоритмами (программирование) начинается именно с реализации этих правил. В языках программирования распределение памяти осуществляется *декларативными операторами* (*операторами описания переменных*). В языке Бейсик не все переменные описываются, обычно описываются только массивы. Но все равно при запуске программы транслятор языка анализирует все идентификаторы в

тексте программы и отводит память под соответствующие переменные.

Третье правило – дискретность. Алгоритм строится из отдельных шагов (действий, операций, команд). Множество шагов, из которых составлен алгоритм, конечно.

Четвертое правило – детерминированность. После каждого шага необходимо указывать, какой шаг выполняется следующим, либо давать команду остановки.

Пятое правило – сходимость (результативность). Алгоритм должен завершать работу после конечного числа шагов. При этом необходимо указать, что считать результатом работы алгоритма.

Итак, алгоритм - неопределяемое понятие теории алгоритмов. Алгоритм каждому определенному набору входных данных ставит в соответствие некоторый набор выходных данных, т.е. вычисляет (реализует) функцию. При рассмотрении конкретных вопросов в теории алгоритмов всегда имеется в виду какая-то конкретная модель алгоритма.

2.2 Виды алгоритмов и их реализация

Алгоритм применительно к вычислительной машине – точное предписание, т.е. набор операций и правил их чередования, при помощи которого, начиная с некоторых исходных данных, можно решить любую задачу фиксированного типа.

Виды алгоритмов как логико-математических средств отражают указанные компоненты человеческой деятельности и тенденции, а сами алгоритмы в зависимости от цели, начальных условий задачи, путей ее решения, определения действий исполнителя подразделяются следующим образом:

- *механические алгоритмы*, или иначе детерминированные, жесткие (например, алгоритм работы машины, двигателя и т.п.). Механические алгоритмы задают определенные действия, обозначая их в единственной и достоверной последовательности, обеспечивая тем самым однозначный требуемый или искомый результат, если выполняются те условия процесса, задачи, для которых разработан алгоритм;
- *гибкие алгоритмы*, например, стохастические, т.е. вероятностные и эвристические.
  - *вероятностные* (*стохастические*) *алгоритмы* дают программу решения задачи несколькими путями или способами, приводящими к вероятному достижению результата;
  - *эвристические алгоритмы* (от греческого слова «эврика») - это такие алгоритмы, в которых достижение конечного результата программы действий однозначно не предопределено, так же как не обозначена вся последовательность действий, не выявлены все действия исполнителя. К эвристическим алгоритмам относят, например, инструкции и предписания. В этих алгоритмах используются универсальные логические процедуры и способы

принятия решений, основанные на аналогиях, ассоциациях и прошлом опыте решения схожих задач;

- *линейные алгоритмы* - наборы команд (указаний), выполняемых последовательно во времени друг за другом;
- *разветвляющиеся алгоритмы* - алгоритмы, содержащие хотя бы одно условие, в результате проверки которого ЭВМ обеспечивает переход на один из двух возможных шагов;
- *циклические алгоритмы* - алгоритмы, предусматривающие многократное повторение одного и того же действия (одних и тех же операций) над новыми исходными данными. К циклическим алгоритмам сводится большинство методов вычислений, перебора вариантов.

Цикл программы - последовательность команд (серия, тело цикла), которая может выполняться многократно (для новых исходных данных) до удовлетворения некоторого условия.

Вспомогательный (*подчиненный*) *алгоритм* (процедура) - алгоритм, ранее разработанный и целиком используемый при алгоритмизации конкретной задачи. В некоторых случаях при наличии одинаковых последовательностей указаний (команд) для различных данных с целью сокращения записи также выделяют вспомогательный алгоритм.

2.3 Методы представления алгоритмов

На практике наиболее распространены следующие формы представления алгоритмов:

- словесная (записи на естественном языке);
- графическая (изображения из графических символов);
- псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
- программная (тексты на языках программирования).

Словесное описание алгоритма.

Данный способ получил значительно меньшее распространение из-за его многословности и отсутствия наглядности.

Рассмотрим пример на алгоритме нахождения максимального из двух значений:

- определим форматы переменных X, Y, M, где X и Y - значения для сравнения, M - переменная для хранения максимального значения;
- получим два значения чисел X и Y для сравнения;
- сравним X и Y;
- если X меньше Y, значит большее число Y;
- поместим в переменную M значение Y;
- если X не меньше (больше) Y, значит большее число X;

- поместим в переменную M значение X.

Словесный способ не имеет широкого распространения по следующим причинам:

- такие описания строго не формализуемы;
- страдают многословностью записей;
- допускают неоднозначность толкования отдельных предписаний.

Структурная (блок-) схема алгоритма

Данный способ оказался очень удобным средством изображения алгоритмов и получил широкое распространение в научной и учебной литературе.

Структурная (блок-, граф-) схема алгоритма - графическое изображение алгоритма в виде схемы связанных между собой с помощью стрелок (линий перехода) блоков - графических символов, каждый из которых соответствует одному шагу алгоритма. Внутри блока дается описание соответствующего действия.

Графическое изображение алгоритма широко используется перед программированием задачи вследствие его наглядности, т.к. зрительное восприятие обычно облегчает процесс написания программы, ее корректировки при возможных ошибках, осмысливание процесса обработки информации.

Можно встретить такое утверждение: «Внешне алгоритм представляет собой схему - набор символов, внутри которых записывается, что вычисляется, что вводится в машину и что выдается на печать и другие средства отображения информации». Здесь форма представления алгоритма смешивается с самим алгоритмом.

Принцип программирования «*сверху вниз*» требует, чтобы блок-схема поэтапно конкретизировалась и каждый блок «расписывался» до элементарных операций. Но такой подход можно осуществить при решении несложных задач. При решении сколько-нибудь серьезной задачи блок-схема «расползется» до такой степени, что ее невозможно будет охватить одним взглядом.

Блок-схемы (структурные схемы) алгоритмов удобно использовать для объяснения работы уже готового алгоритма, при этом в качестве блоков берутся действительно блоки алгоритма, работа которых не требует пояснений. Блок-схема алгоритма должна служить для упрощения изображения алгоритма, а не для усложнения.

Блок «процесс» применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.

Блок «ветвление» используется для обозначения переходов управления по условию. В каждом блоке «ветвление» должны быть указаны вопрос, условие или сравнение, которые он определяет.

Блок «модификация» используется для организации циклических конструкций. (Слово «модификация» означает видоизменение, преобразование.) Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.

Блок «предопределенный процесс» используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

В таблице 2.1 приведены наиболее часто употребляемые символы.

Таблица 2.1 – Типовые обозначения структурной схемы

Название символа	Обозначение и пример заполнения	Пояснение
Процесс	$x = \frac{a-b}{\sin(l)}$	Вычислительное действие или последовательность действий
Ветвление	да, $a < b$, нет	Проверка условий
Модификация	i = 2, 50, 2	Начало цикла
Предопределенный процесс	Расчет параметров	Вычисления по подпрограмме, стандартной подпрограмме
Ввод–вывод	Ввод *a, b, c*	Ввод-вывод в общем виде
Пуск–останов	Начало	Начало, конец алгоритма, вход и выход в подпрограмму
Документ	Печать *b, c*	Вывод результатов на печать

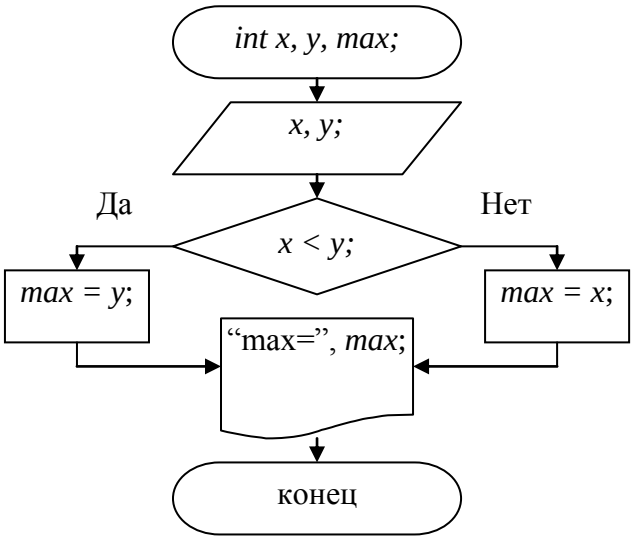


Рисунок 2.1 – Структурная схема алгоритма нахождения максимального из двух значений

Псевдокод

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов. Он занимает промежуточное место между естественным и формальным языками.

С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы на нем могут записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя. Однако в псевдокоде обычно имеются некоторые конструкции, присущие формальным языкам, что облегчает переход от записи на псевдокоде к записи алгоритма на формальном языке. В частности, в псевдокоде так же, как и в формальных языках, есть служебные слова, смысл которых определен раз и навсегда. Они выделяются в печатном тексте жирным шрифтом, а в рукописном тексте подчеркиваются. Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций.

Программное представление алгоритма

При записи алгоритма в словесной форме, в виде блок-схемы или на псевдокоде допускается определенный произвол при изображении команд. Вместе с тем такая запись точна настолько, что позволяет человеку понять суть дела и исполнить алгоритм.

Однако на практике в качестве исполнителей алгоритмов используются компьютеры, поэтому алгоритм, предназначенный для исполнения на

компьютере, должен быть записан на «понятном» ему языке. И здесь на первый план выдвигается необходимость точной записи команд, не оставляющей места для произвольного толкования их исполнителем.

Язык для записи алгоритмов должен быть формализован. Такой язык принято называть языком программирования, а запись алгоритма на этом языке - программой для компьютера.

2.4 Порядок разработки иерархической схемы реализации алгоритмов

К основным методам структурного программирования относится, прежде всего, отказ от бессистемного употребления оператора непосредственного перехода *GOTO* и преимущественное использование других структурированных операторов, методы нисходящего проектирования разработки программы, идеи пошаговой детализации и некоторые другие соглашения, касающиеся дисциплины программирования.

Всякая программа, в соответствии со структурным подходом к программированию, может быть построена с использованием только трех основных типов блоков.

1. Функциональный блок, который на блок-схеме изображается в виде прямоугольников с одним входом и одним выходом.

Функциональному блоку в языках программирования соответствуют операторы ввода и вывода или любой оператор присваивания. В виде функционального блока может быть изображена любая последовательность операторов, выполняющихся один за другим, имеющая один вход и один выход.

2. Условная конструкция. Этот блок включает проверку некоторого логического условия (P), в зависимости от которого выполняется либо один (S1), либо другой (S2) операторы.

3. Блок обобщенного цикла. Этот блок обеспечивает многократное повторение выполнения оператора S пока выполнено логическое условие P.

При конструировании программы с использованием рассмотренных типов блоков эти блоки образуют линейную цепочку так, что выход одного блока подсоединяется к входу следующего. Таким образом, программа имеет линейную структуру, причем порядок следования блоков соответствует порядку, в котором они выполняются.

Такая структура значительно облегчает чтение и понимание программы, а также упрощает доказательство ее правильности. Так как линейная цепочка блоков может быть сведена к одному блоку, то любая программа может, в конечном итоге, рассматриваться как единый функциональный блок с один входом и одним выходом.

При проектировании и написании программы нужно выполнить обратное преобразование, то есть этот блок разбить на последовательность подблоков, затем каждый подблок разбить на последовательность более мелких блоков до тех пор, пока не будут получены «атомарные» блоки,

рассмотренных выше типов. Такой метод конструирования программы принято называть *нисходящим* («сверху вниз»).

При нисходящем методе конструирования алгоритма и программы первоначально рассматривается вся задача в целом. На каждом последующем этапе задача разбивается на более мелкие подзадачи, каждая подзадача, в конечном итоге на еще более мелкие подзадачи и так до тех пор, пока не будут получены такие подзадачи, которые легко кодируются на выбранном языке программирования. При этом на каждом шаге уточняются все новые и новые детали («пошаговая детализация»).

В процессе нисходящего проектирования сохраняется строгая дисциплина программирования, то есть разбиение на подзадачи осуществляется путем применения только рассмотренных типов конструкций (функциональный блок, условная конструкция, обобщенный цикл), поэтому, в конечном итоге, получается хорошо структурированная программа.

2.5 Нормальный алгоритм Маркова

По аналогии с интуитивным определением алгоритма определим понятие «алгоритм в алфавите *A*»:

I. *Алгоритмом в алфавите A* называется всякое общепонятное точное предписание, определяющее потенциально осуществимый процесс над словами из А, допускающий любое слово в качестве исходного и последовательно определяющий новые слова в этом алфавите.

Алгоритм *применим к* некоторому слову *P*, если, отправляясь от этого слова и действуя согласно предписаний, мы получим в конце концов новое слово *Q*, на котором процесс оборвется. Будем тогда говорить, что алгоритм *перерабатывает P в Q*.

Например, следующее предписание удовлетворяет нашему определению. *Перепиши заданное слово, начиная с конца. Полученное слово есть результат. Остановка.* Этот алгоритм представляет собой совершенно точное предписание, применимое к любому слову.

Однако определение **I.** слишком широко: уточнение понятия «алгоритм в алфавите *A*» связано с использованием аппарата подстановок, т.е. с построением ассоциативного исчисления.

II. Будем считать, что алгоритм в алфавите *A* задается в виде некоторой *системы допустимых подстановок, дополненной* общепонятным, *точным предписанием о том, в каком порядке и как нужно применять допустимые подстановки, и когда наступает остановка.*

Так как алфавит и система допустимых подстановок задают ассоциативное исчисление, которому, как мы знаем, можно поставить в соответствие бесконечный лабиринт, то во вторую составную часть этого определения алгоритма (т.е. общепонятное и точное предписание о том, как пользоваться подстановками) можно понимать как точное предписание о способе движения в бесконечном лабиринте.

Приведем пример алгоритма в алфавите *A* в смысле **II.**

Пусть алфавит A содержит три буквы: $A = \{a, b, c\}$, а алгоритм определен с помощью системы подстановок

$$\left.\begin{array}{l} cd - cc \\ cca - ad \\ ab - bca \end{array}\right\}$$

и следующего указания о способе применения этих подстановок: исходя из произвольного слова P, следует просмотреть схему подстановок в том порядке, в каком они выписаны, разыскивая первую формулу, левая часть которой входит в P. Если же такой формулы не найдется, то процесс обрывается сразу же. В противном случае берется первая из найденных формул и делается подстановка ее правой части вместо *первого* вхождения ее левой части в слово P, что дает новое слово P_1, в алфавите A. Затем слово P_1 играет роль P, т.е. вновь берется в качестве исходного, и процесс повторяется. Остановка наступает в том случае, если на n-м шаге получено слово P_n, в которое не входит ни одна из левых частей формул схемы подстановок.

Итак, схема подстановок вместе с указанием, как ими пользоваться, определяет алгоритм в алфавите A. Этот алгоритм перерабатывает слово *babaac* в слово *bbcaac* (применена 3-я формула), слово *cbacacb* - последовательно в слова *ccacacd, ccacacc, abcacc* и, наконец, в *bcacacc*, на котором процесс обрывается; слово *bcacabc* порождает бесконечно повторяющуюся последовательность слов *bcacabc*: *bcacbcac*, *bcaccac*, *bcacabc* и т.д., т.е. остановка не наступит, и, следовательно, к слову *bcacabc* наш алгоритм неприменим.

Наш алгоритм несколько напоминает такой способ задания движения в бесконечном лабиринте: *выйдя на какую-либо площадку, иди в первый коридор направо и т.д. Остановка наступит, когда придешь в тупик.* Ясно, что здесь также может быть три случая: отходя от данной площадки, мы можем либо попасть в тупик (сравним слово *cbacacb),* либо бесконечно кружиться по циклу (сравним слово *bcacabc*), либо двигаться бесконечно долго, не попадая в циклы.

С первого взгляда можно решить, что определение алгоритма в смысле **II** уже, чем определение в смысле **I**. Однако оказывается, что на деле такого сужения нет, ибо для любого известного алгоритма в смысле **I** может быть построен эквивалентный ему алгоритм в смысле **II**. Это, конечно, не доказательство того, что определения **I** и **II** равносильны: такого доказательства не может существовать вообще в силу неточности и расплывчатости обоих определений (везде фигурируют слова «общепонятное, точное предписание»).

Во всяком случае, очевидно, что определение **II** - шаг вперед по пути уточнения понятия «алгоритм».

Определим теперь понятие эквивалентности алгоритмов: *два алгоритма A_1 и A_2 в некотором алфавите называются эквивалентными, если*

области *их применимости совпадают, и результаты переработки ими любого слова из общей области применимости также совпадают.* Иначе говоря, если алгоритм A_1 применим к некоторому слову P, то и A_2 должен быть применим к этому слову, и наоборот;

Оба алгоритма должны перерабатывать слово P в некоторое слово Q. Если же один из алгоритмов неприменим к некоторому слову B, то и другой алгоритм должен быть неприменим к нему.

Для того, чтобы дать точное математическое определение алгоритма, потребовалось сделать еще один шаг по пути дальнейшего уточнения. Этот шаг был сделан А. А. Марковым. Построенный им *нормальный алгоритм,* так же, как и алгоритм в смысле определения **II**, выражен в терминах системы подстановок; однако вместо расплывчатого «общепонятного указания» о том, как пользоваться подстановками, Марков дал стандартные, раз и навсегда определенные точные указания о порядке использования подстановок. Определение нормального алгоритма Маркова таково:

Задается алфавит A и фиксируется схема подстановок. Алгоритм предписывает, исходя из произвольного слова P в алфавите A, просмотреть формулы подстановок в том порядке, в каком они заданы в схеме, разыскивая формулу с левой частью, входящей в P. Если такой формулы не найдется, то процесс обрывается. В противном случае берется первая из таких формул и делается подстановка ее правой части вместо первого вхождения ее левой части в P, что дает новое слово P_1, в алфавите A. После только что выполненного первого шага процесса приступают ко второму его шагу, отличающемуся от первого только тем. что роль P играет P_1. Далее делают аналогичный третий шаг и т.д. до тех пор, пока не придется оборвать процесс. Оборваться же он может лишь двумя способами: во-первых, когда мы получим такое слово P_n, что ни одна из левых частей формул схемы подстановок не будет в него входить; во-вторых, когда при получении слова P_n нам придется применить последнюю формулу. В обоих этих случаях мы считаем, что наш алгоритм перерабатывает слово P в слово P_n.

Как мы видим, приведенный в определении **II** пример является примером «почти нормального» алгоритма. Вся разница состоит в том, что там остановка наступает лишь в одном случае, когда ни одна из подстановок неприменима, а здесь остановка может наступать в двух случаях.

Различные нормальные алгоритмы отличаются друг от друга лишь алфавитами и системами допустимых подстановок. Чтобы задать какой-либо нормальный алгоритм достаточно задать алфавит и систему подстановок.

Понятие алгоритма в некотором алфавите было уточнено Марковым следующим образом: *всякий алгоритм в алфавите А эквивалентен некоторому нормальному алгоритму в этом же алфавите.*

Это утверждение является гипотезой; оно не может быть строго доказано, так как с одной стороны фигурирует расплывчатое понятие «всякий алгоритм», а с другой стороны - точное понятие «нормальный алгоритм». На это утверждение можно смотреть как на закон, который не доказан, но который проверен и подтвержден всем нашим опытом.

В пользу высказанной гипотезы говорит и тот факт, что никому еще не удалось сформулировать пример такого алгоритма в алфавите *A,* для которого нельзя было бы построить эквивалентный ему нормальный алгоритм.

Возвращаясь к содержанию последних абзацев предыдущего параграфа, естественно рассматривать алгоритм Маркова как удобную «стандартную форму» для задания *любого* алгоритма, т. е. предположить, что вообще любой алгоритм может быть задан в форме нормального алгоритма Маркова. Разумеется, это не более чем гипотеза, и притом значительно менее ясная, чем гипотеза Маркова, о которой шла речь выше, так как она не может даже быть высказана в точных терминах, но интуитивный смысл ее очевиден.

Как только мы примем эту гипотезу, сразу становится ясным один из путей, каким можно строго проводить доказательство алгоритмической неразрешимости того или иного круга проблем.

Например, чтобы доказать алгоритмическую неразрешимость проблемы слов, т.е. доказать, что не существует алгоритма, позволяющего для любого ассоциативного исчисления и для произвольных заданных слов *P* и *Q,* ответить на вопрос: эквивалентны ли эти два слова, достаточно было построить пример ассоциативного исчисления и доказать, что для этого исчисления *не существует нормального алгоритма, распознающего эквивалентность слов.*

Впервые такие примеры были построены А.А. Марковым в 1946 г. и Э. Постом в 1947 г. После этого стало ясно, что не существует алгоритма для распознавания эквивалентности слов в любом ассоциативном исчислении. Примеры ассоциативных исчислений, приведенные Марковым и Постом, были весьма громоздкими, насчитывали сотни допустимых подстановок. Позже ленинградский математик Г.С. Цейтин привел пример ассоциативного исчисления, насчитывающего всего лишь семь допустимых подстановок, в котором проблема эквивалентности слов также алгоритмически неразрешима.

В качестве иллюстрации приведем одну схему доказательства алгоритмической неразрешимости.

Пусть в некотором алфавите $A=\{a_1, a_2, \ldots a_n\}$ задан с помощью системы подстановок нормальный алгоритм *U.* В записи этого алгоритма, кроме букв алфавита *A,* содержатся символы $\rightarrow$ и , . Приписав этим символам новые буквы a_{n+1} и a_{n+2} мы получим возможность изображать алгоритм *U словом* в расширенном алфавите $A=\{a_1, a_2, \ldots a_{n+2}\}$. Применим теперь алгоритм *U* к слову, которое его изображает.

Если алгоритм *U* перерабатывает это слово в некоторое иное слово, после чего наступает остановка, то это означает, что алгоритм *U* применим к собственной записи. Такой алгоритм назовем *самоприменимым.* В противном случае алгоритм будем называть *несамоприменимым.* Естественно, возникает задача распознавания самоприменимости: по записи данного алгоритма определить, самоприменим этот алгоритм или нет.

Решение этой задачи мыслится в виде построения некоторого нормального алгоритма *V.* который, будучи применен ко всякой записи самоприменимого алгоритма *U,* перерабатывает эту запись в некоторое слово *M,* а примененный ко всякой записи несамоприменимого алгоритма *U,* перерабатывает эту запись в некоторое иное слово *L*. В этом случае по результату применения распознающего алгоритма *V* мы могли бы узнать, является ли данный алгоритм *U* самоприменимым или нет.

Доказано, что такого нормального алгоритма *V* не существует: тем самым доказано, что проблема распознавания самоприменимости алгоритмически неразрешима. Доказательство проводится от противного. Допустим, что нормальный алгоритм *V* распознавания построен и перерабатывает всякую запись самоприменимого алгоритма в слово *M*, а всякую запись несамоприменимого алгоритма в слово *L*.

Тогда, путем некоторого изменения системы подстановок алгоритма *V* можно построить иной алгоритм $\tilde{U}$, который всякую записи несамоприменимого алгоритма по-прежнему перерабатывает в слово *L*, а ко всякой записи самоприменимого алгоритма неприменим (остановка никогда не наступает). Итак, $\tilde{U}$ применим ко всякой записи несамоприменимого алгоритма (перерабатывая ее в слово *L*) и неприменим к записи самоприменимых алгоритмов (остановка не наступает). Однако это приводит к противоречию. Действительно:

1. Пусть $\tilde{U}$ самоприменим, т.е. он применим к собственной записи в виде слова (остановка наступает). Но это свидетельствует как раз о том, что $\tilde{U}$ несамоприменим.

2. Пусть $\tilde{U}$ несамоприменим, тогда он применим к своей записи (так как он применим к любой записи несамоприменимого алгоритма): но это означает как раз, что $\tilde{U}$ самоприменим.

Полученное противоречие доказывает алгоритмическую неразрешимость проблемы распознавания самоприменимости.

Итак, решая какую-либо задачу, приходится считаться с тем, что алгоритм для ее решения может существовать, а может и не существовать. Поэтому одновременно с поиском нужного алгоритма приходится направлять усилия и на доказательство его несуществования.

Отметим еще, что несуществование алгоритма для решения того или иного класса задач не означает неразрешимости вообще; это означает лишь, что рассматриваемый класс задач настолько широк, что единого эффективного метода для решения всех задач не существует. Так, хотя в ассоциативном исчислении Г.С. Цейтина общая проблема распознавания эквивалентности слов алгоритмически неразрешима, тем не менее для конкретных пар слов мы обычно тем или иным способом можем сделать вывод об их эквивалентности или неэквивалентности.

До уточнения понятия «алгоритм» в математике было две точки зрения:

1) *Все проблемы, для решения которых, пока не удалось найти алгоритм, алгоритмически разрешимы, но искомый алгоритм еще не найден,*

потому что не хватает средств в современной математике для его построения. Иначе говоря, в наше время ситуация в ряде проблем похожа на ситуацию, когда пытались найти площадь круга с помощью циркуля и линейки или решить уравнение *n*-ой степени в радикалах. Естественно, что решения найдено не было, так как для этого использовались недостаточные средства. Может быть и нам для решения проблем, которые мы называем алгоритмически неразрешимыми, просто не хватает средств современной математики, и построение искомых алгоритмов дело завтрашнего дня?

2) *Существуют классы задач, для решения которых не существует алгоритма вообще.* Иначе говоря, есть такие проблемы, которые нельзя решать механически с помощью формальных рассуждений и вычислений и которые требуют творческого мышления.

Утверждение это тем более сильно, что оно имеет характер прогноза на все будущие времена, применительно ко всем будущим средствам. Не тратьте сил зря на поиски несуществующих алгоритмов!

Но как могли бы сторонники второго взгляда доказать несуществование какого-либо алгоритма? До тех пор, пока в определении алгоритма так или иначе фигурируют слова «общепонятное точное предписание», о таком доказательстве нельзя и думать, так как невозможно вести доказательство путем перебора всех «общепонятных точных предписаний» и показа того, что ни одно из них не годится.

Поэтому само существование второй точки зрения возможно только благодаря наличию смелых гипотез с существовании «стандартных форм» для любого алгоритма, например, нормального алгоритма Маркова, т.е. гипотез, делающих возможным сформулировать снятия «алгоритм» и «алгоритмически неразрешимая проблема» в точных терминах.

2.6 Языки программирования

Компьютер не может работать без программного обеспечения - то есть, программ, которые управляют различными аппаратными компонентами компьютера в последовательности, определенной пользователем. Простой заменой программы компьютер может быть настроен так, чтобы работать различными способами. Другими словами, программное обеспечение обеспечивает компьютер универсальной вычислительной способностью.

Рассмотрим простую вычислительную задачу: прибавить число *X*, хранящееся в памяти по адресу 8, к числу *Y* в памяти с адресом 10 и затем сохранить результат в памяти с адресом 5. Это задание может быть записано как следующая программа, состоящая только из трех команд, хотя программа в общем случае может состоять из любого числа команд:

ЗАГРУЗИТЬ 8
ДОБАВИТЬ 10
ХРАНИТЬ 5

Первая команда, ЗАГРУЗИТЬ 8, считывает число *X* из памяти по адресу 8, и помещает его в арифметико-логическое устройство (АЛУ), удаляя

любое число, ранее помещенное там. Вторая команда добавляет число *Y*, хранящееся в памяти по адресу 10, к *X* в АЛУ. Третья команда сохраняет сумму, содержащуюся в АЛУ, в памяти с адресом 5. Числа *X* и *Y* не должны быть упомянуты явно в этой программе, потому что принято, что они были сохранены в соответствующих ячейках памяти предыдущими командами, которые здесь не показываются. Первая часть каждой команды, типа ЗАГРУЗИТЬ, называется *кодом операции*, а вторая часть, типа 8, называется *операндом*.

Некоторые вычислительные среды имеют команды, в каждой из которых содержатся два или три операнда. Например - ДОБАВИТЬ 8 10 означает сложение чисел, хранящихся в памяти по адресам 8 и 10. Команды в программе называются *кодом*.

Каждая программа составляется по определенным правилам, которые называются *языком программирования*, которых существует огромное количество. Компьютер может непосредственно понимать только машинный язык, который является уникальным для каждого отдельного компьютера (т.е. архитектуры). Таким образом, команда ЗАГРУЗИТЬ 8, в предыдущем примере, могла бы быть записана, как 010100001000 в машинном коде, где 0101 в начале означает операцию ЗАГРУЗИТЬ и где последующие символы 00001000 означают десятичное число 8.

Хотя компьютеры легко работают с машинным языком, такой код затруднителен для записи и чтения людьми. Поэтому программисты используют языки, которые являются удобочитаемыми для них. Например, мнемонический код операции ЗАГРУЗИТЬ на человеческом языке - гораздо проще для понимания, чем 0101 на машинном языке. Такой читаемый человеком язык должен транслироваться на машинный язык программами специального назначения, называемыми *языковыми процессорами*.

Языки программирования могут быть классифицированы с точки зрения удобочитаемости. Языки, удобочитаемость которых является близкой к таковой для машинных языков, называют *языками низкого уровня*, например, *ассемблеры*. Те языки, чья удобочитаемость является близкой к человеческим языкам, называются *языками высокого уровня*. Бейсик и Паскаль - примеры языков высокого уровня. Программы, написанные на языках низкого уровня, обычно могут быть выполнены быстрее и требовать меньшего пространства памяти в компьютере, чем написанные на языках высокого уровня.

КОБОЛ, язык высокого уровня, который широко использовался для деловых прикладных приложений, имеет структуру, подобную структуре английского языка и его словарю. Рассмотрим следующий пример программы на COBOL:

```
IF ITEM -- COUNT = 0
THEN
NEXT SENTENCE
ELSE
```

```
COMPUTE AVERAGE--PRICE =
AVERAGE -- PRICE / ITEM -- COUNT.
MOVE AVERAGE -- PRICE TO
PRINT--AVERAGE--PRICE
PERFORM 300 – PRINT – DETAIL -- LINE.
```

Слово COMPUTE, например, легко будет понято пользователями, знакомым с английским языком, без проблем, поэтому, например, COMPUTE, может быть сокращено до COMP или даже до еще более короткого слова, требующего меньшее количество памяти.

Языки программирования классифицируются с различных точек зрения, таких, как основные области их приложений или способы решения проблем. Фортран и Алгол классифицированы как языки программирования для научных или проектных приложений, в то время как Кобол и Лисп подходят для деловых проблем. Некоторые языки, типа PL/1, Бейсик, С и Паскаль, являются универсальными языками. Например, и научные, и проектные программы, и языковые процессоры, и деловые программы, как правило, написаны на различных диалектах языка С (например, С++, С# и т.д.).

Команды, как описано ранее, являются шагами программ, написанных в трансляторах или машинных языках. Каждый шаг языка высокого уровня называется *инструкцией*. Каждая команда определяет операцию АЛУ, памяти или других аппаратных средств. Каждая инструкция, однако, определяет вычислительную задачу, которая не может быть непосредственно связана с аппаратными действиями и которая в общем случае соответствует многим командам, когда транслируется в ассемблер или машинный язык.

Языковые процессоры

Языковые процессоры подразделяются на трансляторы и интерпретаторы. *Транслятор* - программа, которая конвертирует программу, написанную на одном языке, в программу на другом языке так, чтобы обе давали идентичные результаты вычислений. В этом случае первичная программа, ее код и язык называют исходной программой, исходным текстом и исходным языком, соответственно. Результаты транслированной версии называют программой-адресатом (объектом, целью), целевым (объектным) кодом и целевым (объектным) языком. Трансляторы далее классифицируются на ассемблеры, компиляторы и препроцессоры.

Интерпретатор преобразовывает каждую инструкцию программы непосредственно в машинный код и немедленно выполняет ее перед переходом к следующей инструкции.

Ассемблеры

Ассемблер транслирует программу, написанную на компоновочном языке (язык ассемблера/assembly language), в машинный код. Каждая команда в программе, написанной на ассемблере, имеет почти взаимно однозначное соответствие командам в машинном коде. Другими словами,

код операции и операнд, из которых состоит каждая инструкция ассемблера, обозначаются читаемым именем (т. е. мнемоническим кодом операции) и десятичным числом, вместо двоичного представления соответствующей машинной команды. LOAD 8 - пример такой команды на ассемблере, соответствующий машинной команде 010100001000.

Таким образом, по сравнению с другими языковыми процессорами, ассемблеры имеют относительно простые структуры. Каждая инструкция на языке высокого уровня имеет намного более сложную связь с машинным кодом, поэтому в ассемблере программы пишутся тогда, когда желательно использовать компьютерные аппаратные средства более эффективно. Ассемблеры, однако, не столь же читаемы, как языки высокого уровня, и, следовательно, сохраняется возможность создания ошибок при увеличении объема программирования.

Компиляторы

Компилятор - транслятор со сложной структурой, который преобразовывает программу, написанную на языке высокого уровня, в машинный код или, в некоторых случаях, в программу на ассемблере.

Препроцессоры

Препроцессор - языковый процессор, который выполняет предварительную трансляцию исходных кодов типа замены алфавитно-цифровых выражений двоичной формой и вставкой определенных файлов, создавая модификацию исходного текста, который должен быть далее обработан транслятором. Препроцессор также удобно использовать, когда новый язык высокого уровня создается путем добавления элементов к существующему языку высокого уровня. Например, программа на C++ может транслироваться в ее эквивалент на C препроцессором.

Интерпретаторы

Интерпретатор выполняет каждую инструкцию программы, поскольку она преобразована в машинный код без создания программы-цели, в то время как ассемблеры и компиляторы производят программы-цели без выполнения их. Интерпретаторы удобны для быстрого нахождения ошибок в программах, но не подходят для больших программ из-за низкого времени интерпретации.

Существует много языков программирования для различных целей. Здесь рассмотрены некоторые из наиболее важных.

Языки Assembler

Хотя компьютеры от одного изготовителя имеют тенденцию использовать один и тот же машинный язык, для машин различных изготовителей это не так. Соответственно, различные компьютеры имеют различные компонующие программы и языки ассемблирования.

В дополнение к преобразованию мнемонического операционного кода

и десятичного операнда каждой команды в машинный код, большинство языков ассемблирования имеют возможности для рационализации программирования, такие, как объединение последовательности отдельных команд в единую псевдокоманду. Из этой псевдокоманды затем генерируется инструмент в машинном коде.

Программирование на ассемблерах требует твердого знания архитектуры вычислительной системы и более трудоемко, чем программирование на языках высокого уровня.

Пример программного кода:

```
.386
.model flat
extrn ExitProcess:PROC
extrn MessageBoxA:PROC

.data
Ttl db "First ASSEMBLER program",0h
Msg db 'Hello, World!!!!',0h

.code
start:
    push 0h
        push offset Msg
        push offset Ttl
        push 0h
        call MessageBoxA
        push 0h
        call ExitProcess
end   start
```

ФОРТРАН

ФОРТРАН (ForTran, транслятор формул) был создан как язык для решения задач числового анализа Джоном В. Бекусом и рядом других специалистов из IBM и был анонсирован в 1957 г. С тех пор он был несколько раз модернизирован. Даже если другие языки, например С, становятся популярными для научных и технических вычислений, ФОРТРАН остается избранным языком для численного анализа. Чтобы расширить его применимость для научных вычислений вне сферы численного анализа, в версии, выпущенной в 1990 г., ФОРТРАН 90 были добавлены средства для обработки структурированных данных, динамическое распределение данных, рекурсивные расчеты и другие возможности. ФОРТРАН в новых диалектах иногда применяется в настоящее время.

Пример программного кода:

```
      EXTERNAL ZN
      WRITE (*,'(A)')
      DATA A/2./, B/3./, C/6./, Xn/-5./, Xk/5./, H/0.5/
      X=Xn
1     Y=ZN(A,B,C,X)
      WRITE(*,2) X,Y
2     FORMAT (1X,'X=',F5.1,2X,'Y=',F6.2)
      X=X+H
```

```
IF (X.LE.Xk) GOTO 1
STOP
END
FUNCTION ZN(A,B,C,X)
IF (-A.LE.X.AND.X.LE.A) P=0
IF (X.GE.B) P=C
IF (-B.GE.X) P=-C
ZN=P
RETURN
END
```

Кобол

Кобол (COmmon Business-Oriented Language - ориентированный на бизнес язык широкого применения) - был наиболее популярным языком в мире бизнеса, включая банки и страховые компании. Пользователи компьютеров и их изготовители объединились с Министерством обороны США, чтобы установить единый язык программирования для деловых приложений и создали Конференцию по Языкам Систем Данных (CODASYL) в 1959 г. CODASYL создал КОБОЛ для достижения двух главных целей: портативность (способность программ к переносу при минимальных изменениях на компьютеры, произведенные различными компаниями) и удобочитаемость (легкость, с которой программы могут читаться, подобно обычным предложениям на английском).

КОБОЛ был пересмотрен несколько раз с 1959 г. Он мог быть более легко понят деловыми людьми, чем, возможно, другие языки; и программы, написанные в КОБОЛЕ, весьма компактны.

Пример программного кода:

```
PROCEDURE DIVISION.
 move 16 to n
 move 0 to i
 move 1 to fact
 perform until i greater than n
  move i to ist
  move fact to factst
  display ist "! = " factst
  add 1 to i
  multiply i by fact
   on size error display "value too big"
  end-multiply
 end-perform.
 stop run.
```

PL/1

PL/1 - комплексный язык, который был предложен SHARE (группа пользователей IBM компьютеров) и IBM в 1963 г. Он был первоначально назван NPL (новый язык программирования), но позже переименован в PL/1. IBM впервые анонсировал описание PL/1 в 1965 г. Американский Национальный Институт Стандартов (ANSI) и другие организации с тех пор исправляли его несколько раз. PL/1 был разработан для научных, проектных и деловых проблем, объединяя элементы ФОРТРАНа, КОБОЛа и АЛГОЛа,

которые в то время были весьма популярными.

Пример программного кода:

```
EXTSUB:PROC(NAM,AG) RETURNS(FIXED BIN(15,0));
DCL NAM CHAR(10);
DCL AG FIXED BIN(15,0);
PUT SKIP LIST('NAME:',NAM);
PUT SKIP LIST('AGE:',AG);
AG=25;
RETURN (AG);
END EXTSUB
```

Бейсик

Бейсик (BASIC - Beginner's All-purpose Symbolic Instruction Code, Универсальная Символическая Система команд для начинающих) - универсальный язык программирования, разработанный Джоном Дж. Кемени и Томасом Е. Куртцем в Дартмаус-колледже (Гановер, США) в середине 1960-х г. Это один из самых простых языков высокого уровня и он относительно легко изучается школьниками и начинающими программистами. Приблизительно с 1980 г. БЕЙСИК стал популярным для использования на персональных компьютерах. Отдельно развивается диалект VBA. VBA - Visual Basic for Applications — немного упрощённая реализация языка программирования Visual Basic, встроенная в линейку продуктов Microsoft Office (включая версии для Mac OS), а также во многие другие программные пакеты, такие как AutoCAD, SolidWorks, CorelDRAW, WordPerfect и ESRI ArcGIS. VBA покрывает и расширяет функциональность ранее использовавшихся специализированных макро-языков, таких как WordBasic.

VBA является интерпретируемым языком. Как и следует из его названия, VBA близок к Visual Basic. VBA, будучи языком, построенным на COM (Component Object Model — объектная модель компонентов), позволяет использовать все доступные в операционной системе COM объекты и компоненты ActiveX. По сути, возможно создание приложения на основе Microsoft Word VBA, использующего только средства Corel Draw.

Пример программного кода:

```
RANDOMIZE (TIMER)
LOCATE 10, 20: PRINT "УГАДАЙ ГДЕ ШАРИК?"
A = 99 * RND(1)
IF A <= 33 THEN B = 1 ELSE IF A > 66 THEN B = 3 ELSE B = 2
LOCATE 11, 20: PRINT " _  _  _ "
LOCATE 12, 20: PRINT "/ \ / \ / \"
LOCATE 13, 20: PRINT " 1 2 3"
INPUT "ВВЕДИ НОМЕР СТАКАНА ", C
IF C = B THEN PRINT "УГАДАЛ! МОЛОДЕЦ!" ELSE PRINT "НЕ УГАДАЛ!"
LOCATE 11, 20: PRINT " "
LOCATE 12, 20: PRINT "\_/ \_/ \_/ "
LOCATE 12, 17 + 4 * B: PRINT "O"
```

Паскаль

Паскаль (Pascal) - язык, разработанный Н. Виртом из Федерального

Института Технологии (Цюрих, Швейцария) в конце 1960-х г.г. Он предназначался в качестве хорошего методического инструмента для систематического изучения программирования и имел быстрые, надежные компиляторы. Начиная с 1974 г. компилятор Паскаля, разработанный Виртом, стал доступен публике и использовался многими университетами.

Паскаль сильно повлиял на многие языки, разработанные позже. Описание языка Паскаль лаконично, что делает его более простым для обучения, чем многие другие языки высокого уровня. Комплексные структуры данных и алгоритмы могут быть описаны кратко на Паскале, и его программы просты для чтения и отладки.

Пример программного кода:

```
write('Введите размерность 1 массива (от 2 до ',Size,'):');
read (n);
until (n>1) and (n<=Size);
Randomize;
a[1]:=Random(Diap);
write ('A= ',a[1],' ');
for i:=2 to n do begin
a[i]:=Random(Diap);{заполнение массива случайными числами}
write (a[i],' '); {вывод элементов массива}
end;
```

C

Хотя C (читается- «Си») рассматривается как язык высокого уровня, он имеет много элементов низкого уровня, типа способности непосредственно обращаться к адресам и битам. C, тем не менее, высоко портативен. Он был развит Деннисом М. Ритчи из лабораторий компании AT&T в 1972 г. Операционная система UNIX была написана почти исключительно на C; предшествующие операционные системы были почти полностью написаны на ассемблере или в машинных кодах. C широко использовался на персональных и больших компьютерах, переродившись со временем в другие свои диалекты.

Пример программного кода:

```
#include <stdio.h>
 int main (void) {
    printf ("Hello, World!\n");
    return 0;
 }
```

Лисп

Лисп (LISP – LISt Processor/списочный процессор) - язык, который является мощным средством скорее для управления списками данных или символов, чем для обработки числовых данных. В этом смысле Лисп уникален. Он требует памяти большой емкости и, так как он обычно обрабатывается интерпретатором, медленно выполняется в программах. Лисп был создан в конце 1950-х г.г. и в начале 1960-х г.г. группой, возглавляемой Джоном Маккарти, впоследствии профессором Института Технологии в Штате Массачусетс. В то время, Лисп радикально отличался от

других языков типа ФОРТРАНа и АЛГОЛа. Отдельные версии были развиты из Лисп 1.5, представленного Маккарти. Лисп, выпущенный в 1984 г., соответствует фактическому стандарту Лисп.

Пример программного кода:

```
(defun queue-empty-p (queue)
  "Return T if QUEUE is empty."
  (check-type queue queue)
  (= (queue-put-ptr queue) (queue-get-ptr queue)))
(defun queue-full-p (queue)
  "Return T if QUEUE is full."
  (check-type queue queue)
  (= (queue-get-ptr queue)
    (queue-next queue (queue-put-ptr queue))))
```

Языки четвертого поколения

Они ближе к человеческому языку, чем другие языки высокого уровня. Языки четвертого поколения предназначены, чтобы быть более простыми для пользователей, чем машинные языки (первое поколение), ассемблеры (второе поколение) и языки высокого уровня (третье поколение). Это проблемно-ориентированные языки, оперирующие конкретными понятиями узкой области, как правило, в такие языки встраивают мощные операторы, позволяющие одной строкой описывать функции, для описания которых языках младших поколений потребовалось бы сотни или даже тысячи строк исходного кода.

К языкам четвертого поколения часто относят: SQL, SGML (HTML, XML), Prolog и многие другие узкоспециализированные декларативные языки. Следует отметить, что ряд языков, которые относят к четвертому поколению, не являются языками программирования как таковыми. Например SQL является языком запросов к базам данных, HTML является языком разметки гипертекста, а не полноценными языками программирования, скорее они выступают своеобразными специализированными дополнениями к языкам программирования. То же самое касается XML.

Основная отличительная особенность языка четвертого поколения: приближение к человеческой речи (декларативные языки). Некоторые языки имеют черты одновременно и третьего и четвертого поколений.

Объектно-ориентированные языки программирования

Объектно-ориентированное программирование использует методику программирования, где программа написана с дискретными объектами, которые сами содержат собрание вычислительных процедур и структур данных. Новые программы могут быть написаны путем сборки группы этих заранее предопределенных, автономных объектов в более короткое время, чем написание полных программ с самого начала. Алгоритмический язык Симула и Smalltalk - примеры ранних объектно-ориентированных языков. С 1990ых годов объектно-ориентированное программирование стало чрезвычайно популярным из-за высокой производительности

программирования. Язык C++, который был создан Бьерном Страуструпом из AT&T в начале 1980-х г.г., и Object-C, который был разработан Брэдом Коксом в 1984 г., являются объектно-ориентированными версиями C, которые стали очень популярны. То же касается и языка Object Pascal.

Delphi

Delphi (Делфи) — это объектно-ориентированный язык программирования. Подразумевает комбинацию нескольких важнейших технологий:

- Высокопроизводительный компилятор в машинный код
- Объектно-ориентированная модель компонент
- Визуальное (а, следовательно, и скоростное) построение приложений из программных прототипов
- Масштабируемые средства для построения баз данных

Изначально язык был предназначен исключительно для разработки приложений Microsoft Windows, затем был реализован также для платформ GNU/Linux (как Kylix), однако после выпуска в 2002 году Kylix 3 его разработка была прекращена, и, вскоре после этого, было объявлено о поддержке Microsoft .NET. При этом высказывались предположения, что эти два факта взаимосвязаны.

Реализация языка Delphi проектом Free Pascal позволяет использовать его для создания приложений для таких платформ, как Mac OS X, Windows CE и Linux.

Пример программного кода:

```
 FileExt := AnsiUpperCase(ExtractFileExt(FileListBox1.Filename));
  if (FileExt = '.BMP') or (FileExt = '.ICO') or (FileExt = '.WMF') or
    (FileExt = '.EMF') then
  begin
    Image1.Picture.LoadFromFile(FileListBox1.Filename);
    Caption := FormCaption + ExtractFilename(FileListBox1.Filename);
    if (FileExt = '.BMP') then
    begin
      Caption := Caption +
        Format(' (%d x %d)', [Image1.Picture.Width, Image1.Picture.Height]);
      ViewForm.Image1.Picture := Image1.Picture;
      ViewForm.Caption := Caption;
      if GlyphCheck.Checked then ViewAsGlyph(FileExt);
    end
    else
      GlyphCheck.Checked := False;
end;
```

C++

C++ - компилируемый статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования как процедурное программирование, модульность, раздельная компиляция, обработка исключений, абстракция данных, типы (объекты), виртуальные функции,

объектно-ориентированное программирование, обобщенное программирование, контейнеры и алгоритмы, сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

Являясь одним из самых популярных языков программирования, C++ широко используется для разработки программного обеспечения. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (например, видеоигры). C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#.

Пример программного кода:

первый вариант синтаксиса

```
#include <stdio.h>
int main()
{
int x, y;
x = 5;
y = 6;
printf( "x: %d\ny: %d\n", x, y );
printf("1 + 2: %d\n", 1 + 2 );
x = 12 / 3;
y = 2 * x;
printf( "x: %d\ny: %d\n", x, y );
return 0;
}
```

второй вариант синтаксиса

```
#include <iostream>
using namespace std;
int x;
int main()
{
        std::cin >> x;
   std::cout << "Вы ввели число:" << x;
        cout << "Hello, world! /n";
   return 0;
}
```

C#

C# (C Sharp, Си шарп) - простой, современный, объектно-ориентированный язык с безопасной системой типов, происходящий от C и C++. C# будет удобен и понятен для программистов, знающих C и C++. C# сочетает продуктивность Visual Basic и мощность C++».

Переняв многое от своих предшественников — языков C++, Java, Delphi, Модула и Smalltalk — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не

поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

Пример программного кода:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace p
{
   class Program
   {
      static void Main(string[] args)
      {
         string str = "FlkjsdfSdsfA";
         Console.WriteLine(str);
         for (int i = 0; i < str.Length; i++)
            if (char.IsUpper(str[i])) Console.WriteLine("{0} в верхнем регистре ", str[i]);
      }
   }
}
```

Java

Java - объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (позже приобретённой компанией Oracle). Дата официального выпуска — 23 мая 1995 года.

Программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинство подобного способа выполнения программ — в полной независимости байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание.

Пример программного кода:

```
import java.io.*;
// This is a simple program called HelloWorld.java
class HelloWorld
  {
public static void main(String args[ ])
    {
System.out.println( Welcome to the world of Java );
    }
  }
```

2.7 Жизненный цикл программного обеспечения

Жизненный цикл программного обеспечения включает в себя шесть этапов:

- анализ требований,
- определение спецификаций,
- проектирование,
- кодирование,
- тестирование,
- сопровождение.

Рассмотрим их.

Анализ требований. При разработке программного обеспечения он исключительно важен. Ошибки, допущенные на этом этапе, даже при условии безупречного выполнения последующих этапов могут привести к тому, что разработанный программный продукт не будет соответствовать требованиям практики, сферы его применения. Для создания конкурентоспособных продуктов в ходе выполнения этого этапа должны быть получены четкие ответы на следующие вопросы:

- Что должна делать программа?
- В чем состоят реальные проблемы, разрешению которых она должна способствовать?
- Что представляют собой входные данные?
- Какими должны быть выходные данные?
- Какими ресурсами располагает проектировщик?

Определение спецификаций. В определенной степени этот этап можно рассматривать как формулировку выводов, следующих из результатов предыдущего этапа. Требования к программе должны быть представлены в виде ряда спецификаций, явно определяющих рабочие характеристики будущей программы. В число таких характеристик могут входить скорость выполнения, объем потребляемой памяти, гибкость применения и др.

Проектирование. На этом этапе создается общая структура программы, которая должна удовлетворять спецификациям; определяются общие принципы управления и взаимодействия между различными компонентами программы.

Кодирование. Заключается в переводе на язык программирования конструкций, записанных на языке проектирования.

Тестирование. На этом этапе производится всесторонняя проверка программ. Тестирование более подробно рассмотрено ниже.

Сопровождение. Это этап эксплуатации системы. Каким бы изощренным не было тестирование программ, к сожалению, в больших программных комплексах чрезвычайно тяжело устранить абсолютно все ошибки. Устранение обнаруженных при эксплуатации ошибок - первейшая задача этого этапа. Однако это далеко не все, что выполняется при сопровождении. Выполняемый в ходе сопровождения анализ опыта

эксплуатации программы позволяет обнаруживать «узкие места» или неудачные проектные решения в тех или иных частях программного комплекса. В результате такого анализа может быть принято решение о проведении работ по совершенствованию разработанной системы. Кроме описанного выше, сопровождение может включать в себя проведение консультаций, обучение пользователей системы, оперативное снабжение пользователей информацией о новых версиях системы и т.п. Качественное проведение этапа сопровождения в большой степени определяет коммерческий успех программного продукта.

Рассмотрим этап тестирования программ более подробно. Существуют три аспекта проверки программы на:

- правильность;
- эффективность реализации;
- вычислительную сложность.

Проверка правильности удостоверяет, что программа делает в точности то, для чего она была предназначена. Математическая безупречность алгоритма не гарантирует правильности его перевода в программу. Аналогично, ни отсутствие диагностических сообщений компилятора, ни разумный вид получаемых результатов не дают достаточной гарантии правильности программы. Как правило, проверка правильности заключается в разработке и проведении набора тестов. Кроме этого, для расчета программ иногда можно сверить получаемые решения с уже известным решением. Таким образом, нельзя дать общего решения для проведения проверки на правильность программы.

Проверка вычислительной сложности, как правило, заключается в экспериментальном анализе сложности алгоритма или экспериментальном сравнении двух алгоритмов и более, решающих одну и ту же задачу.

Проверка эффективности реализации направлена на отыскание способа заставить правильную программу работать быстрее или расходовать меньше памяти. Чтобы улучшить программу, пересматриваются результаты реализации в процессе построения алгоритма. Не рассматривая все возможные варианты и направления оптимизации программ, приведем здесь некоторые полезные способы, направленные на увеличение скорости выполнения программ.

Первый способ основан на следующем правиле. Сложение и вычитание выполняются быстрее, чем умножение и деление. Целочисленная арифметика быстрее арифметики вещественных чисел. Таким образом, $X+X$ лучше, чем $2X$, а $i+0{,}5$j хуже, чем $(2i+j)\times 0{,}5$ или $(i+i+j)\times 0{,}5$. При выполнении операций над целыми числами следует помнить, что благодаря применению двоичной системы счисления умножение на числа, кратные двум, можно заменить соответствующим количеством сдвигов влево. Второй способ заключается в удалении избыточных вычислений.

Третий способ проверки эффективности реализации основан на способности некоторых компиляторов строить коды для вычисления

логических выражений так, что вычисления прекращаются, если результат становится очевидным. Например, в выражении *A* or *B* or *C*, если *A* имеет значение «истина», то переменные *B* и *C* уже не проверяются. Таким образом, можно сэкономить время, разместив переменные *A, B, C* так, чтобы первой стояла переменная, которая вероятнее всего будет истинной, а последней та, которая реже всего принимает истинное значение.

Четвертый прием - исключение циклов.

Пятый прием - развертывание циклов.

Это далеко не полный перечень способов оптимизации. Здесь приведены лишь самые очевидные из них. Следует, кроме того, заметить, что не всегда стоит увлекаться погоней за быстродействием, так как при этом чаще всего ухудшается удобочитаемость программ. В том случае, когда выигрыш получается «мизерный», вряд ли стоит предпочитать его ясности и читабельности программы.

2.8 Основы технологии разработки программ

Необходимо кратко коснуться части процесса проектирования программы [19]:

- язык программирования, как таковой, не имеет отношения к процессу написания программ (и можно даже прийти к выводу о практической бесполезности многочисленной литературы по системам программирования);
- любая технология программирования имеет отношение прежде всего к формальной стороне проектирования. Так, структурное программирование предполагает последовательное движение от внешних программных конструкций к внутренним, но откуда берутся эти конструкции – не говорит;
- программы не создаются из набора заготовок путем их механического или случайного соединения. Даже если известны составные части программы, в какой последовательности их соединять?

Образная и логическая стороны программирования

Поднятые выше вопросы останутся без ответа, если ограничиться формальной стороной процесса т.е. исключительно технологией программирования. Но в самом начале на любом шаге проектирования программы имеют место элементы образного представления программы. Попробуем взглянуть на все это с максимально широких позиций.

Мыслительная и творческая деятельность человека имеет две формы: образно-художественную и формально-логическую. Есть основания связывать их с различием функционирования правого (образного) и левого (логического) полушария головного мозга. Инженерному и научному подходам ближе формально-логическое мышление: здесь человек оперирует ограниченным числом четко отделенных друг от друга объектов на основе строгих формальных или логических законов. Наоборот, образ является

неделимой сущностью и всегда воспринимается как единое целое. Манипулирование над образом состоит в непосредственном «видении» процесса его изменения. К образному мышлению относятся также такие понятия как интуиция, опыт, аналогия и т.п.. Хотя образное мышление традиционно воспринимают как составляющую художественного творчества и необходимый элемент гуманитарных наук, в технических и технологических отраслях он тоже является важным элементом. Вообще, без него невозможно получение нового знания, ибо любая формальная система допускаем манипулирование только в собственных рамках, что, естественно, ограничивает ее сверху.

В процессе разработки программы образная сторона заключается в представлении программы в виде целостной «движущейся картинки», из которой очевидно, как выполняется процесс, приводящий к результату. Словесные формулировки алгоритма типа «переместить выбранный элемент к концу массива» уже сочетают в себе образное и формально-логическое (алгоритмическое) описание действия. Следовательно, программирование – это движение в направлении от образной модели к словесным формулировкам составляющих действий, а уже затем к формальной их записи на языке программирования[19].

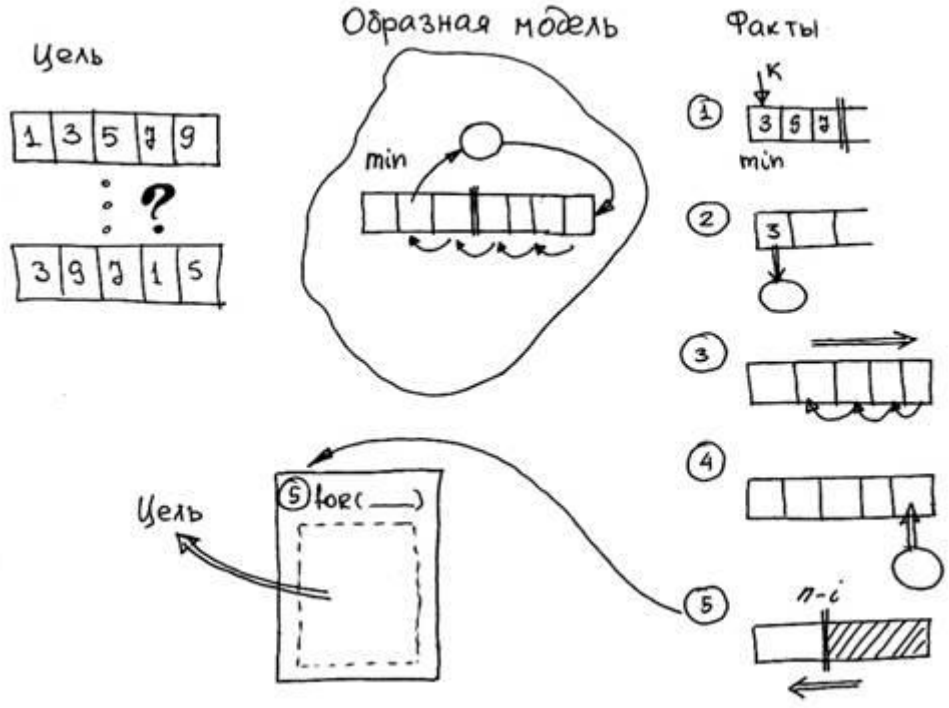


Рисунок 2.2 – Образное программирование на бумаге – залог успеха проектирования программы

Попробуем более подробно определить составляющие этого процесса.

1. Результат работы программы. Целью выполнения любой программы является получение результата, а результат – это данные с определенными свойствами. Например, целью программы сортировки является создание

последовательности из имеющихся данных, расположенных в порядке возрастания. Точно так же любой промежуточный шаг программы имеет свою цель: получить данные с нужными свойствами в нужном месте. Как правило, постановка задачи начинается с формулировки результата. Для простого случая он может быть задан в самой постановке задачи, а в сложных случаях, особенно если речь идет о заказчике, не очень хорошо владеющим формальной стороной проблемы, его требования к исполнителю (программисту) могут звучать как «Сделайте мне красиво».

2. Образная модель программы. Формальное проектирование программы не продвинется ни на шаг, если программист «не видит», как это происходит. То есть первоначально действующая модель программы должна присутствовать в голове. Понятно, что к формальной логике это не имеет никакого отношения. Это – область образного мышления, (правого полушария). Изобразительные средства здесь уместны любые – словесные, графические. Здесь работают интуиция, аналогия, фантазия и другие элементы творческого процесса. На этом уровне справедлив тезис, что программирование – это искусство. Насколько подробно программист «видит» модель в движении и насколько он способен описать это словами – настолько он близок к следующему этапу проектирования.

3. Факты, касающиеся программы. Формальная сторона проектирования начинается с перечисления фактов, касающихся образной модели программы. К таковым относятся: переменные и их смысловая интерпретация, известные программные решения и соответствующие им стандартные программные контексты. Сразу же надо заметить, что речь идет не об окончательных фрагментах программы, а о фрагментах, которые могут войти в готовую программу. Иногда при их включении потребуется доопределить некоторые параметры (например, границы выполнения цикла, которые не видны на этапе сбора фактов). Иногда они могут быть эквивалентно преобразованы (то есть иметь другой синтаксис). Умение видеть в алгоритме известные частные решения тоже приобретается с опытом: для этого и нужно учиться «читать» программы.

По поводу этого пункта следует указать на важнейшую роль в этом процессе естественного языка (речи). Человек мыслит образами, а речь его представляет собой уже формально-логическую структуру. Она состоит из отдельных единиц – слов, их взаимосвязь – синтаксис и смысл – семантика, также подчинены определенным законам. Понятно, что естественный язык многообразен, словами в нем могут быть выражены как образные (например, поэзия), так и формально-логические системы (например, математика). Для программирования как раз важно, что переход от образной модели к ее известным составляющим состоит в словесных формулировках. Если Вы не способны сформулировать свои образные представления о процессе работы программы в виде словесных формулировок отдельных действий, дальше двигаться бесполезно. Поэтому вопреки науки о том, что этого пока нельзя сделать, программирование уже давно ведется на естественном языке!

4. Выстраивание программы из набора фактов. Эта часть процесса программирования вызывает наибольшие затруднения, ибо здесь начинается то, что извне обычно и воспринимается как «программирование»: написание текста программы. Особенность заключается в том, что обычно фрагменты взаимосвязаны друг с другом прямо по структуре алгоритма или косвенно через данные. Различие подходов состоит в том, в какой последовательности в программу включаются фрагменты (по отношению к гипотетической готовой программе), с какой стороны начать этот процесс и в каком направлении двигаться[19].

«Историческое» проектирование соответствует естественному ходу рассуждений по линии наименьшего сопротивления. Программист просто записывает очередной оператор, который по его мнению должен выполняться программой в процессе ее работы. Ошибочность такого принципа состоит в том, что текст программы и последовательность ее выполнения - это не одно и то же и расхождение между ними рано или поздно обнаружится. Хорошо, если это случится, когда большая часть программы уже написана, и проблема будет скорректирована несколькими «заплатками» в виде операторов *goto*. Заметим, что «историческим» подходом грешат не только программы, но и любые другие структурированные тексты (например, курсовая работа, диплом, диссертация), если автор не уделяет должного внимания логике их построения.

Восходящее проектирование – проектирование программы «изнутри», от самой внутренней конструкции к внешней. Привлекательность этого подхода обусловлена тем, что внутренние конструкции программы - это частности, которые всегда более «на виду», чем внешние конструкции, реализующие обобщенные действия. Частности составляют большую часть фактов в образной модели программы и, что самое ценное, могут быть непосредственно записаны на языке программирования. Поэтому программа в процессе своего написания не нуждается, как и в «историческом» подходе, в иных средствах описания, кроме самого языка программирования. Недостатки тоже очевидны:

- не факт, что программу удастся «свести» в единое целое, особенно сложную;

- поскольку параметры внутренних конструкций могут зависеть от внешних (например, диапазон поиска минимального значения во внутреннем цикле зависит от шага внешнего цикла), то внутренние конструкции - не есть «истины в последней инстанции» и по мере написания программы тоже должны корректироваться.

Нисходящее (структурное) проектирование - проектирование программы, начиная с самой внешней ее конструкции. Самое трудное, но самое правильное движение в направлении от общего к частному. Первая трудность заключается в неочевидности выбора самой внешней (общей, объемлющей) конструкции – частности всегда виднее. Вторая, менее

очевидная: ненаписанная часть программы (внутреннее содержимое конструкции) также должна быть сформулирована в общем виде, т.е. словесно. Отсюда следует, что нисходящее проектирование должно сочетать в тексте программы формальное (то есть записанное на языке программирования) и неформальное (то есть словесное или даже образное) представление.

«Грязное» программирование – заключается в создании макета, воспроизводящего основные свойства проектируемой программы. В дальнейшем все изменения/дополнения сохраняют эти первоначальные соотношения.

5. Последовательное приближение к результату. Сложную программу не всегда удается спроектировать за один этап. Цикл «результат – образная модель – факты – выстраивание программы» может повторяться. При выстраивании программы может оказаться, что ненаписанная часть программы нуждается в дополнительном осмыслении, начиная с образной модели. При этом само направление выстраивания программы, т.е. собственно технология программирования, имеют большое значение: она в большей или меньшей степени гарантируется правильность и неприкосновенность уже написанного.

Кратко рассмотрим упомянутые ранее методики[19].

«Историческое» проектирование – естественный подход к проектированию. Первое, что приходит в голову при старте разработки программы – записывать последовательность действий в проектируемом алгоритме в том порядке, в котором они будут выполняться в программе.

«Историческому» принципу проектирования также наиболее соответствует структурная схема алгоритма. К этому есть несколько причин:

- «рисование» блок-схемы идет в виде волны: разработчик каждый раз решает в контексте текущего состояния, что же делать дальше, при наличии условий процесс проектирования становится параллельным: каждую ветку можно вести независимо от другой;
- циклы возникают естественным образом, при возникновении ощущения, что следующая последовательность действий уже встречалось в программе, следует сделать переход к этой части программы (обратная связь цикла). При этом необходимо проверить и другие составляющие возникающего цикла: условия завершения (или продолжения), переход к следующему шагу. Возможно, что придется переделать уже «нарисованный» фрагмент – он был спроектирован без учета возникшего цикла.

Структурное программирование – от общего к частному.

Технология структурного программирования в самой краткой формулировке есть нисходящее проектирование, т.е. выстраивание текста программы, точнее алгоритмической компоненты, от общего к частному, от внешней конструкции к внутренней. Естественно, что надо знать, из чего выстраивать. В идеале у опытного программиста действительно очередная

нужная конструкция появляется «из головы». Но это не значит, что он не имеет общего плана действий и обобщенного представления процесса, который реализуется проектируемой программой.

1. Исходным состоянием процесса проектирования является более или менее точная формулировка цели алгоритма, или результата, который должен быть получен при его выполнении. Формулировка, само собой, производится на естественном языке.

2. Создается образная модель происходящего процесса, используются графические и какие угодно способы представления, образные «картинки», позволяющие лучше понять выполнение алгоритма в динамике;

3. Выполняется сбор фактов, касающихся любых характеристик алгоритма, и попытка их представления средствами языка. Такими фактами является наличие определенных переменных и их «смысл», а также соответствующих им программных контекстов. Понятно, что не все факты удастся сразу выразить в виде фрагментов программы, но они должны быть сформулированы хотя бы на естественном языке;

4. В образной модели выделяется наиболее существенная часть – «главное звено», для которой подбирается наиболее точная словесная формулировка;

5. Производится определение переменных, необходимых для формального представления данного шага алгоритма и формулируется их «смысл»;

6. Выбирается одна из конструкций - простая последовательность действий, условная конструкция или цикл. Составные части выбранной формальной конструкции (например, условие, заголовок цикла) должны быть переписаны в словесной формулировке в виде цели или результата, которые должны давать эти части алгоритма.

7. Для оставшихся неформализованных частей алгоритма (в словесной формулировке) - перечисленная последовательность действий повторяется. Обычно разработка образного представления программы опережает ее «выстраивание», поэтому следующим этапом для неформализованной части алгоритма может быть п.4 (в лучшем случае, при его проработке в образной модели) или п.1-3. В любом случае для вложенных конструкций мы возвращаемся на предыдущие этапы проектирования[19].

Неправильное программирование – грязное.

Под «грязным» программированием обычно понимается написание программы, грубо воспроизводящей требуемое поведение. Такая программа может быть быстро разработана и отлажена, а затем использована для уяснения последующих шагов, либо для наложения «заплаток» с целью получения требуемого результата. Хотя этот подход крайне некрасив с точки зрения технологии проектирования, но он может быть оправдан при следующих условиях:

- «грязная» программа воспроизводит требуемое поведение на самом верхнем уровне;

- в дальнейшем в нее могут встраиваться контексты и фрагменты, не меняющие ее поведения, но конкретизирующие ее в нужном направлении.

Наконец, необходимо сказать несколько слов о «среде обитания» программы. Каждая конструкция языка не просто встраивается в программу, а определяет свойства используемых ею данных, «смысл» переменных, которые появились в программе одновременно с ней. Поэтому при использовании исключительно вложенных конструкций мы получим в каждой точке программы определенный набор выполняемых условий, своего рода «среду обитания» алгоритма. Эти переменные являются исходными данными для очередного шага детализации алгоритма.

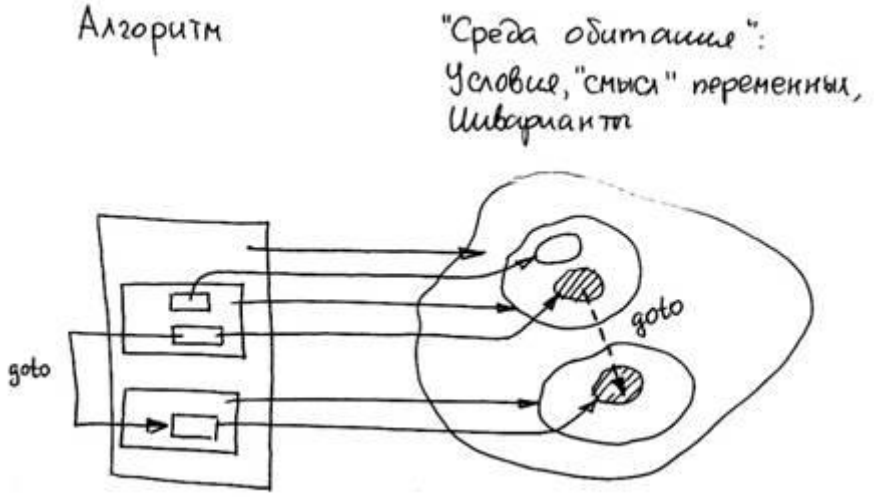


Рисунок 2.3 – «Среда обитания» программы

Почему крайне не рекомендуется использовать при программировании оператор *goto* (оператор безусловного перехода)? Нисходящее пошаговое проектирование исключает использование оператора *goto*, более того, запрещает его применение как нарушающего структуру программы. *Goto* страшен не тем, что «неправильно» связывает разные части алгоритма, а в том, что переводит алгоритм из одних «условий обитания» в другие: в точке перехода они составлены без учета того, что управление будет передано в точку программы «не по правилам».

Тем не менее, есть допустимые случаи использования оператора *goto*. Чрезвычайными обстоятельствами, вынуждающими прибегнуть к помощи оператора *goto*, являются глобальные нарушения логики выполнения программы, например грубые неисправимые ошибки во входных данных. В таких случаях делается переход из нескольких вложенных конструкций либо в конец программы, либо к повторению некоторой ее части. В других обстоятельствах его использование свидетельствует скорее о неправильном проектировании структуры программы - наличии неявных условных или циклических конструкций. Приведем пример достаточно разумного использования оператора *goto*.

начало цикла

...

...

...

ЕСЛИ *** GOTO метка_конец // Сразу уйти в конец программы

...

метка_конец:

...

Контрольные вопросы

1. Перечислите известные вам свойства алгоритма.
2. Перечислите известные вам виды алгоритмов.
3. Какие способы представления алгоритмов вы знаете?
4. Что такое структурная схема алгоритма и из каких элементов она состоит?
5. Как происходит разработка иерархической схемы реализации алгоритма?
6. Что вы знаете о нормальном алгоритме Маркова, в чем его суть?
7. Перечислите несколько классификаций языков программирования.
8. В чем отличия поколений языков программирования?
9. Сформулируйте последовательность этапов жизненного цикла программного обеспечения.
10. Расскажите об основных принципах проектирования программы

3. МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

3.1 Понятие дискретного автомата

В технике термином «*автомат*» пользуются для обозначения системы механизмов и устройств, в которой процессы получения преобразования, передачи и использования энергии, материалов и информации, необходимые для выполнения ее функций, осуществляются без непосредственного участия человека. К системам такого типа относятся: станки-автоматы, фасовочные автоматы, автоматы для съемки и изготовления фотографий, торговые автоматы и многое др.

В кибернетику, однако, вошел и прочно в ней укрепился термин «*дискретный автомат*» или кратко просто «*автомат*» для обозначения гораздо более абстрактного понятия, а именно - модели, обладающей следующими особенностями:

а) на входы модели в каждый из дискретных моментов времени t_1, t_2,… поступают m входных величин $x_1,x_2,\ldots x_m$. каждая из которых может принимать конечное число фиксированных значений из входного алфавита X;

б) на выходах модели можно наблюдать n выходных величин $y_1,\ldots y_n$ каждая из которых может принимать конечное число фиксированных значений из выходного алфавита Y;

в) в каждый момент времени модель может находиться в одном из состояний $z_1,z_2,\ldots z_n$;

г) состояние модели в каждый момент времени определяется входной величиной x в этот момент и состоянием z в предыдущий момент времени;

д) модель осуществляет преобразование ситуации на входе $x=\{x_1,x_2,\ldots,x_m\}$ в ситуацию на выходе $y=\{y_1,y_2,\ldots,y_n\}$ зависимости от ее состояния в предыдущий момент времени.

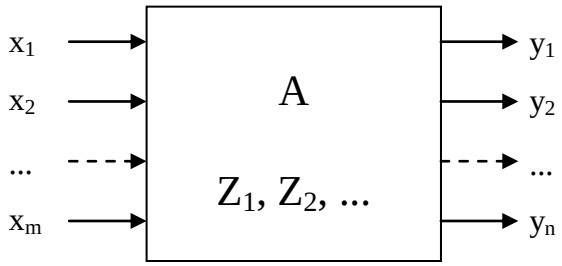


Рисунок 3.1 – Дискретный автомат

Такая модель (рисунок 3.1) удобна для описания многих кибернетических систем.

Автоматы, у которых ситуация y на выходах однозначно определяется ситуацией x на входах, мы будем относить к классу автоматов *без памяти.* Автоматы, у которых y зависит не только от значения x в данный момент, но и от состояния модели z, определяемого значениями x в предыдущие моменты времени, относятся к классу *автоматов с конечной памятью.*

Мы ограничимся рассмотрением лишь простейших из дискретных автоматов, входной и выходной алфавиты которых состоят всего из двух букв: 0 и 1. Это оправдывается тем что, как оказывается в теории автоматов, автоматы с такими "бедными" алфавитами способны решать такие же задачи, как и автоматы с любыми другими алфавитами.

Теория дискретных автоматов приобрела большое значение для решения некоторых фундаментальных проблем информатики, которые связаны с принципиальными возможностями переработки информации в ИС.

Логический автомат

Преобразования входных величин в выходные, осуществляемые дискретными автоматами без памяти, работающими в двухбуквенном алфавите, эквивалентны преобразованиям, совершаемым в формальной логике. Поэтому мы будем называть их *логическими автоматами,* а функции, описывающие преобразования, выполняемые логическими автоматами, - *логическими функциями.* Математическим аппаратом, используемым для решения задач анализа и синтеза логических автоматов, является *алгебра логики.* Первый вариант алгебры логики был разработан английским ученым Джорджем Булем в 1843 г., вследствие чего она часто называется *булевой алгеброй.*

Каждый выход из логического автомата может принимать значение 0 или 1 в зависимости от значений входных переменных *x.* Определим число всех возможных логических функций преобразования x_i в y_i, если число входных величин равно *m,* каждая из них может принимать значение 0 или 1. Для этого расположим все входные величины в ряд x_1, x_2, …, x_m и будем рассматривать их как разряды двоичного числа. Ясно, что число *r* различных сочетаний значений входных величин равно числу различных двоичных чисел, содержащих *r* разрядов, откуда следует, что $r=2^m$. Но каждой из *r* ситуаций на входе может соответствовать одно из двух значений выхода 0 или 1. Поэтому общее число *N* всех различных логических функций для логического автомата с *m* двоичными входами равно

$$N = 2^r = 2^{(2^m)}. \qquad (3.1)$$

Логические функции образуются из некоторых элементарных логических функций. Мы будем пользоваться тремя элементарными логическими функциями:

1. $\bar{x}$ - *отрицание, инверсия x* (читается «не *x*»). Функция отрицания означает, что $x=0$, если $x=1$; и $x=1$, если $x=0$.

2. $x_1 \wedge x_2$ (x_1 & x_2) - *логическое умножение* или *конъюнкция* (читается «x_1 и x_2»). Функция логического умножения означает, что его результат равен единице только тогда, когда $x_1=1$ и $x_2=1$, и равен нулю во всех остальных случаях.

3. $x_1 \vee x_2$ ($x_1 + x_2$) - *логическое сложение* или *дизъюнкция* (читается «x_1 или x_2»). Функция логического сложения означает, что его результат равен

нулю только тогда, когда x_1=0 и x_2=0 , и равен единице во всех остальных случаях.

Логические функции могут задаваться таблицами, в которых указывается значение функции y (индекс i будем опускать) для всех сочетаний аргументов x. В таблице 3.1 приведены значения двух элементарных логических функций от двух аргументов: x_1 и x_2. Эту таблицу нужно читать по строкам: «если x_1 = ..., а x_2 = ..., то x_1 и x_2 =..., а x_1 или x_2 = ...». Логические функции широко используются в теории нейронных сетей и входят в математический аппарат, применяемый при исследованиях процессов переработки информации мозгом.

Таблица 3.1 – Задание логических функций таблицей

Функция	x_1x_2				Примечание
	00	01	10	11	
f_0	0	0	0	0	f_0 – абсолютная ложь
f_1	0	0	0	1	$x_1 \wedge x_2$ (конъюнкция)
f_2	0	0	1	0	$x_1 \bar{x}_2$ (запрет x_2)
f_3	0	0	1	1	$x_1 \bar{x}_2 \vee x_1x_2$ (переменная x_1)
f_4	0	1	0	0	$\bar{x}_1x_2$ (запрет x_1)
f_5	0	1	0	1	$\bar{x}_1 x_2 \vee x_1 x_2$ (переменная x_2)
f_6	0	1	1	0	$x_1 \oplus x_2$ (сложение по модулю 2)
f_7	0	1	1	1	$x_1 \vee x_2$ (дизъюнкция)
f_8	1	0	0	0	$x_1 \downarrow x_2$ (функция Пирса или "стрелка Пирса")
f_9	1	0	0	1	$x_1 \equiv x_2$ (равнозначность)
f_{10}	1	0	1	0	$\bar{x}_1 \bar{x}_2 \vee x_1 \bar{x}_2$ (переменная $\bar{x}_2$)
f_{11}	1	0	1	1	$x_2 \rightarrow x_1$ (импликация)
f_{12}	1	1	0	0	$\bar{x}_1 \bar{x}_2 \vee \bar{x}_1x_2$ (переменная $\bar{x}_1$)
f_{13}	1	1	0	1	$x_1 \rightarrow x_2$ (импликация)
f_{14}	1	1	1	0	x_1/x_2 (функция Шеффера или "штрих Шеффера")
f_{15}	1	1	1	1	f_1 – абсолютная истина

Из элементарных логических функций можно составлять логические функции, описывающие свойства различных логических автоматов.

Логические элементы выполняют преобразования информации, которые описываются логическими функциями (таблица 3.1). Условные графические обозначения (УГО) элементов И, ИЛИ и НЕ на функциональных схемах приведены на рисунке 3.2.

Условное графическое обозначение элемента выполняют в виде прямоугольника, в левом верхнем углу которого указывают обозначение выполняемой функции. Входы элемента обозначают линиями слева, выходы – справа. Инверсный выход обозначается кружком (рисунок 3.2*в*)

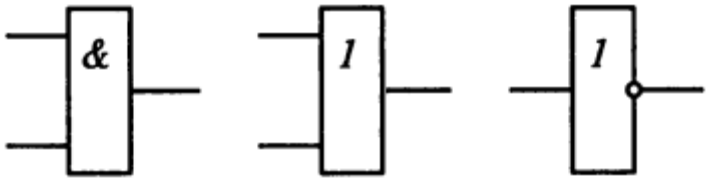


Рисунок 3.2 – Стандартные логические элементы
а – элемент И, *б* – элемент ИЛИ, *в* – элемент НЕ

Элементы И, ИЛИ и НЕ составляют функционально полную систему, т.е. из них можно составить любую комбинационную (логическую) схему. Логические элементы, выполненные в виде интегральных схем, обычно реализуют логические функции И—НЕ (штрих Шеффера) или ИЛИ—НЕ (стрелка Пирса). Каждый из этих элементов обладает свойством функциональной полноты. Обозначения элементов И—НЕ и ИЛИ—НЕ на функциональных схемах, а также примеры их реализации приведены на рисунке 3.3.

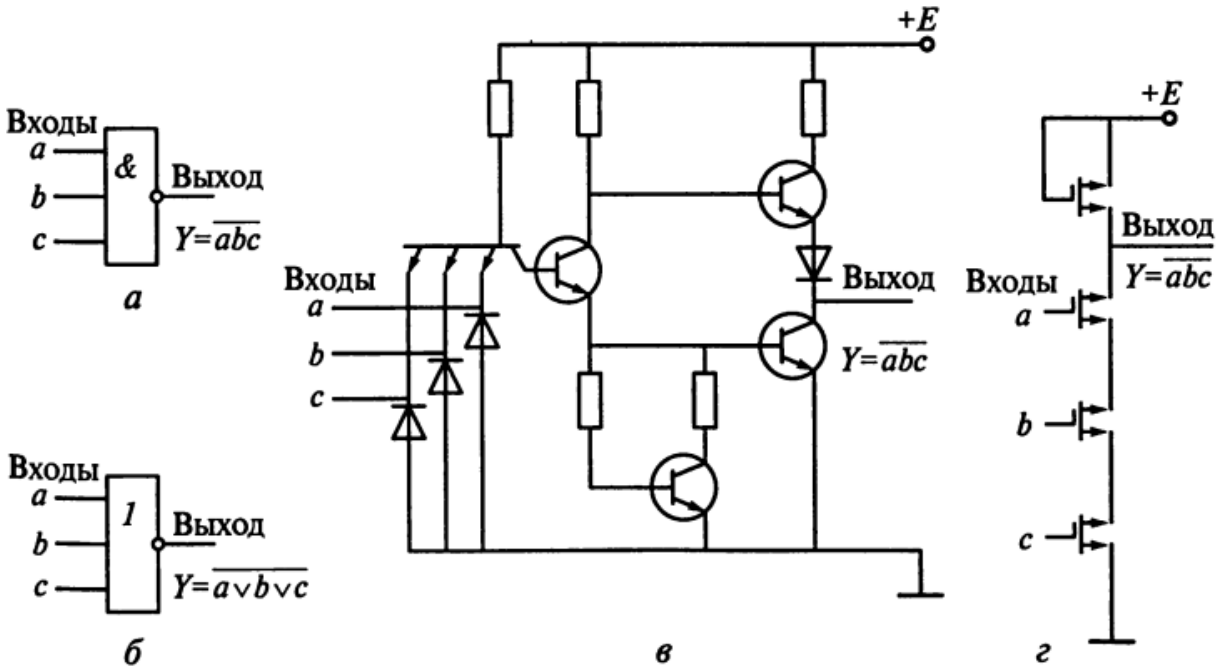


Рисунок 3.3 – Логические элементы И-НЕ, ИЛИ-НЕ
а – элемент И-НЕ на три входа, *б* – элемент ИЛИ-НЕ на три входа, *в* – принципиальная электрическая схема И-НЕ транзисторно-транзисторной логики, *г* – принципиальная электрическая схема И-НЕ на МОП-транзисторах

Элементы И на два входа часто используют в качестве ключей (вентилей), которые управляют передачей данных между двумя схемами (рисунок 3.4).

При передаче одноразрядных данных (рисунок 3.4*а*) один вход элемента И является информационным, а второй — управляющим. На информационный вход поступают одноразрядные данные D от источника А. Если на управляющем входе сигнал «Передать» равен «0», то на приемник В поступает сигнал «0» независимо от значения данных D. Если сигнал «Передать» равен «1», то сигнал на выходе элемента И совпадает с информационным сигналом и на вход приемника поступают данные D.

При передаче многоразрядных данных (рисунок 3.4*б*) каждый разряд данных проходит через отдельный элемент И, а сигнал «Передать» подается одновременно на управляющие входы всех ключей.

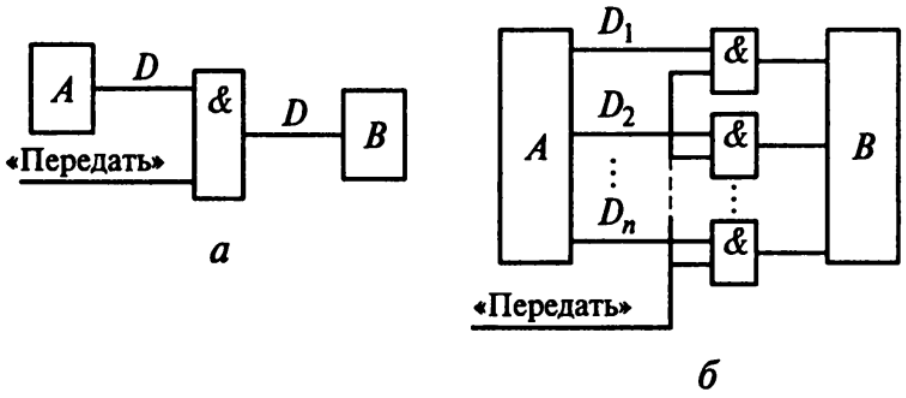


Рисунок 3.4 – Схемы передачи данных
а – одноразрядных, *б* – многоразрядных

Автомат с конечной памятью. Триггеры

При изучении автоматов с конечной памятью обычно интересуются только установившимися состояниями, которые они принимают через достаточно большое время после изменения входных воздействий. Процессы перехода системы из одного установившегося состояния в другое здесь полагаются протекающими достаточно быстро по сравнению с интервалами времени между изменениями входных воздействий. Поэтому поведение автомата с конечной памятью удобно рассматривать в дискретные моменты времени t_1, t_2, ..., отделенные друг от друга интервалами Δt. При этом мы будем полагать, что и выходные воздействия могут изменяться только в моменты t_1, t_2, ..., которые называются *тактами.*

В соответствии с определением выход автомата с конечной памятью в j-й такт зависит от состояния автомата в (j-1)-й такт и состояния входов в j-й такт. Поэтому переходы такого автомата из одного состояния в другое, в общем виде, описываются выражениями

$$\begin{cases} y^j = F(z^{j-1}, x^j), \\ z^j = G(z^{j-1}, x^j), \end{cases} \tag{3.2}$$

где y^j- выход автомата в j-й такт зависит от состояния автомата в (j-1)-й такт, z^{j-1} - состояние автомата в (j-1)-й такт.

$$x^j = \{x_1^j, x_2^j, ..., x_m^j\} \tag{3.3}$$

x^j - вход автомата в j-й такт, F и G - некоторые логические функции состояния выхода и входа.

Для того, чтобы автомат осуществлял преобразование (3.2), необходимо, чтобы он, кроме элементов, реализующих логические функции, содержал также *элемент задержки,* выход которого определяется значением его состояния в предыдущий такт, т. е. элемент, выход которого y связан с входом x выражением

$$y^j = f(z^{j-1}) \tag{3.4}$$

или, в частности,

$$y^j = z^{j-1}. \tag{3.5}$$

Элемент задержки должен обладать памятью, в нем должен сохраняться след предыдущего состояния, ибо иначе его состояние не могло бы зависеть от предыдущего состояния.

Одним из распространенных дискретных элементов, обладающих памятью, является *триггер,* представляющий собой устройство с двумя устойчивыми состояниями. Это устройство может переходить из одного состояния в другое под воздействием сигнала управления.

Рассмотрим в качестве примера автомата с конечной памятью схему электронного счетчика, применяемого в цифровых вычислительных устройствах (рисунок 3.5). Задача этой схемы состоит в подсчете количества импульсов, поступивших на ее вход, т.е. в преобразовании количества импульсов в двоичный код числа, выражающего это количество.

Для этой цели образуем цепь из триггеров, показанную на рисунке. Здесь выход каждого предыдущего триггера соединен с входом последующего. Пусть сначала все триггеры находятся в нулевом состоянии, т.е. напряжение на их выходах равно U_0. При поступлении первого импульса на вход триггера T_1 на его выходе появится напряжение U_1, и на входе триггера T_2, положительный импульс напряжения, на который он не реагирует. Второй импульс заставит T_1 вернуться в нулевое состояние, в результате чего напряжение на его выходе изменит свое значение с U_1 на U_0 , что вызовет отрицательный импульс на входе T_2 и его переход в единичное состояние. Таким образом, T_1 будет изменять свое состояние после каждого входного импульса, T_2 после каждого второго импульса, T_3 - после каждого четвертого и т.д., T_k - после каждого 2^{k-1} импульса на входе схемы. Если теперь мы будем состояние каждого триггера рассматривать как значение соответствующего разряда двоичного числа, то состояние всей цепи из r триггеров будет представлять собой число (в двоичной системе счисления) импульсов, поступивших на вход схемы.

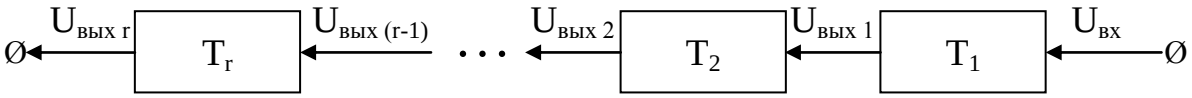


Рисунок 3.5 – Счетчик импульсов на триґгерах

Емкость этой схемы - максимальное число R импульсов, которые могут быть ею сосчитаны, определяется числом r триггеров и равно максимальному двоичному числу, состоящему из r разрядов, а именно $R=2^r$.

Все современные серии цифровых микросхем, как правило включают различные типы триггеров, представляющих устройство с двумя устойчивыми состояниями, содержащее бистабильный запоминающий элемент (ячейку) и схему управления.

Основу триггера составляет бистабильная ячейка, имеющая два устойчивых состояния. Бистабильные ячейки могут быть построены на двух логических элементах И—НЕ или ИЛИ—НЕ, соединенных перекрестными связями (рисунок 3.6).

Существование двух устойчивых состояний бистабильной ячейки объясняется наличием в ее схеме обратных связей, позволяющих сигналу с выхода элемента поступать на его же вход через второй элемент. Так, если на рисунке 3.6*а*, сигнал на верхнем выходе равен«1» и на оба входа подается сигнал «0», то сигнал «1» с выхода элемента 1 поступает на вход элемента 2 и формирует на его выходе сигнал «0». Этот сигнал поступает на вход элемента 1 и поддерживает такое состояние схемы, делая его устойчивым. В этом состоянии на выходе элемента 1 сигнал равен «1», а на выходе элемента 2 сигнал равен «0».

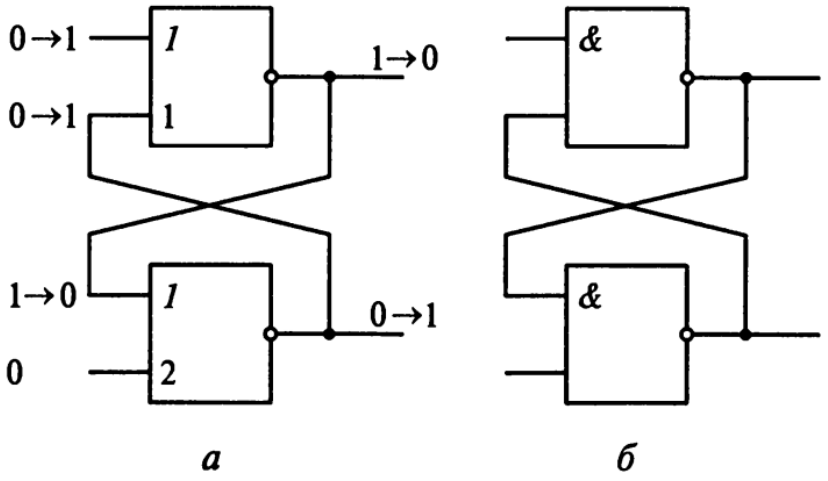


Рисунок 3.6 – Бистабильная ячейка
а – на элементах ИЛИ-НЕ, *б* – на элементах И-НЕ

Для изменения состояния схемы необходимо подать на верхний вход элемента 1 сигнал «1». При этом на выходе элемента 1 сигнал становится равным «0». Тогда на выходе элемента 2 формируется сигнал «1», который вместе с единичным входным сигналом устанавливает выходной сигнал элемента 1 равным «0». Такое состояние схемы также является устойчивым и после того, как сигнал на входе элемента 1 станет равным «0». В этом состоянии на выходе элемента 1 сигнал равен «0», а на выходе элемента 2 — «1». При поступлении сигнала «1» на вход элемента 2 происходит возвращение схемы в начальное состояние. Таким образом, схема имеет два устойчивых

состояния, которые можно устанавливать подачей сигнала «1» на вход элемента 1 или 2. Аналогично работает бистабильная ячейка на элементах И—НЕ (рисунок 3.6б).

Кроме бистабильной ячейки в состав триггера входит схема управления (рисунок 3.7). Схема управления — это комбинационная схема, при помощи которой осуществляется запись информации в триггер (изменение состояний триггера). Конкретный вид схемы управления зависит от типа триггера.

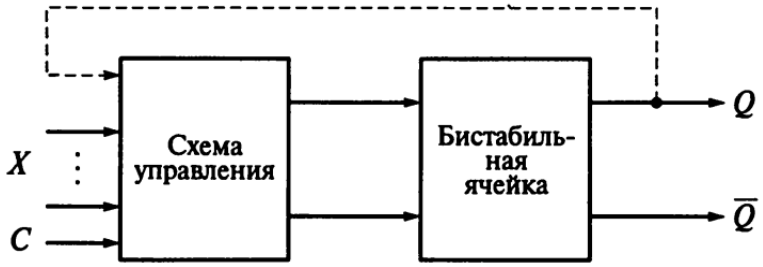


Рисунок 3.7 – Общая структура триггера

Триггер имеет два выхода: прямой и инверсный (Q и $\overline{Q}$). Сигналы на выходах триггера всегда имеют различные значения. Если на прямом выходе сигнал равен «1», то на инверсном — «0» и наоборот. Состояние триггера определяется значением сигнала на прямом выходе (Q). Если сигнал на прямом выходе равен «1», то триггер находится в состоянии «1». Можно также сказать, что состояние триггера — это информация, записанная в триггере. Таким образом, если триггер находится в состоянии «1», то в нем записана единица.

Входы триггеров, как и сигналы, подаваемые на них делятся на информационные и вспомогательные. Информационные сигналы через соответствующие входы управляют состоянием триггера. Сигналы на вспомогательных входах служат для предварительной установки триггера в заданное состояние и его синхронизации. Вспомогательные входы могут при необходимости выполнить роль информационных. По способу приема информации триггеры подразделяют тактируемые (синхронные) и нетактируемые (асинхронные) триггеры. Изменение состояния нетактируемого (асинхронного) триггера происходит сразу же после соответствующего изменения потенциалов на его управляющих входах.

В тактируемом (синхронном) триггере изменение состояния может произойти только в момент присутствия соответствующего сигнала на тактовом входе. Если сигнал синхронизации равен «0», состояние синхронного триггера не изменится при любых значениях сигналов на его входах.

Тактирование может осуществляться импульсом (потенциалом) или фронтом (перепадом потенциала). В первом случае сигналы на управляющих входах оказывают влияние на состояние триггера только при разрешающем потенциале на тактовом входе. Во втором случае воздействие управляющих сигналов проявляется только в момент перехода единица - нуль или нуль -

единица на тактовом входе. Существуют также универсальные триггеры, которые могут работать как в тактируемом, так и в нетактируемом режиме.

Основные типы триггеров в интегральном исполнении носят следующие названия: *D*-триггеры, *T*-триггеры, *RS*-триггеры и *JK*-триггеры.

Асинхронный *RS*-триггер. Он имеет два информационных входа — *R* и *S*. Вход *S* (set) используется для установки триггера в состояние «1», а вход *R* (reset) — в состояние «0», поэтому *RS*-триггер называют триггером с установочными входами.

Работу триггера описывает таблица переходов (таблица 3.2*а*).

Таблица 3.2 – Переходы RS-триггера

(а)

Входы		Состояния	
R	*S*	0	1
0	0	0	1
0	1	1	1
1	0	0	0
1	1	—	—

(б)

Входы		Текущее состояние	Следующее состояние
R	*S*	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	Запрещенная комбинация
1	1	1	

Входами служат значения входных сигналов *R* и *S*, а также значения состояний триггера в текущий момент времени (Q_t). В таблице переходов приведены значения состояний триггера в следующий момент времени (Q_{t+1}). Переходы триггера из одного состояния в другое происходят, если на вход *R* или *S* подается сигнал «1».

При $R = 0$ и $S = 0$ состояние триггера не меняется. Такой режим называется **режимом хранения**. В случае если $R = 0$ и $S = 1$ триггер переходит в состояние «1» независимо от того, в каком состоянии он находился до изменения входных сигналов. При $R = 1$ и $S = 0$ триггер переходит в состояние «0». Таким образом, для записи «1» в *RS*-триггер необходимо подать на его входы сигналы $R = 0$ и $S = 1$, для записи «0» — сигналы $R = 1$ и $S = 0$. Комбинация сигналов $R = 1$ и $S = 1$ является запрещенной, состояние триггера при этом не определено. Таким образом, логика переходов *RS*-триггера аналогична логике работы электрического клавишного выключателя с двумя положениями: «Вкл» и «Выкл».

Таблица переходов триггера может быть интерпретирована как таблица истинности комбинационной схемы, в которой значения сигналов на входах

R_t, S_t и значение текущего состояния Q_t можно рассматривать как логические переменные, а Q_{t+1} как логическую функцию (таблица 3.2*б*).

RS-триггер может быть построен на различных логических элементах. Функциональная схема асинхронного *RS*-триггера, построенного на элементах И-НЕ и ИЛИ-НЕ, а также его условное графическое обозначение (УГО) показаны на рисунке 3.8.

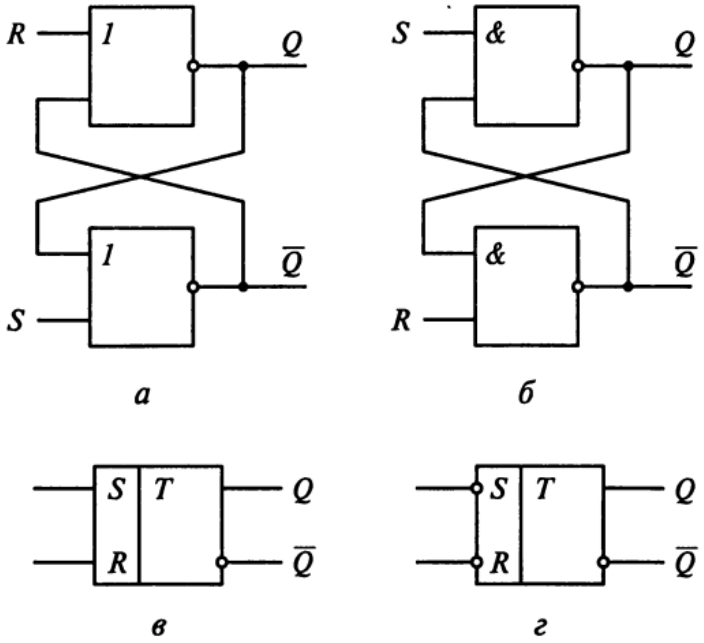


Рисунок 3.8 – Асинхронный RS-триггер
а – на элементах ИЛИ-НЕ, *б* – на элементах И-НЕ, *в* – УГО асинхронного *RS*-триггера с прямыми входами, *г* – УГО асинхронного *RS*-триггера с инверсными входами

Условное графическое обозначение триггера кроме основного поля включает в себя дополнительное. В основном поле указывается назначение элемента (триггер), а в дополнительном — обозначение входов, т. е. тип триггера. Инверсный выход триггера отмечается кружком. Инверсными могут быть и входные сигналы. Так, при построении *RS*-триггера на элементах И—НЕ действующими (активными) являются инверсные значения входных сигналов *R* и *S* (сигналы «0»). Это означает, что для переключения триггера из одного состояния в другое нужно подать сигнал «0» на соответствующий вход триггера (*R* или *S*). Инверсные входы на схемах также отмечаются кружком. Заметим, что таблицы переходов триггеров обычно приводятся для прямых значений входных сигналов. Асинхронный *RS*-триггер представляет собой бистабильную ячейку, поэтому он используется как основа при построении практически всех триггеров.

Синхронный *RS*-триггер. Этот триггер имеет дополнительно вход *C*, на который поступают синхросигналы(такты). Информационные сигналы *R* и *S* могут изменять состояние триггера только при значении синхросигнала *C*=1. Таблица переходов синхронного *RS*-триггера состоит из двух частей.

Первая часть таблицы описывает переходы триггера при $C = 1$ и совпадает с таблицей переходов асинхронного триггера (таблица 3.2*а*). Когда $C = 0$, триггер не меняет своего состояния при любой комбинации сигналов на информационных входах и логика его переходов может быть описана таблицей 3.3.

Отметим, что при $C = 0$ разрешенными являются любые комбинации входных сигналов, в том числе $R = 1$, $S = 1$.

Таблица 3.3 – Переходы синхронного *RS*-триггера

Входы			Состояния	
C	*R*	*S*	0	1
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1

На рисунке 3.9 приведены функциональные схемы синхронных RS-триггеров, реализованных на элементах И—НЕ и И—ИЛИ—НЕ, и их условное графическое обозначение. Кроме основных входов *R* и *S* там показаны дополнительные входы R_1 и S_1 которые являются асинхронными. При подаче сигналов на них состояние триггера может изменяться независимо от значения сигнала *C*. Следует отметить, что в каждый момент времени можно управлять переходами триггера только с помощью синхронных или асинхронных входов.

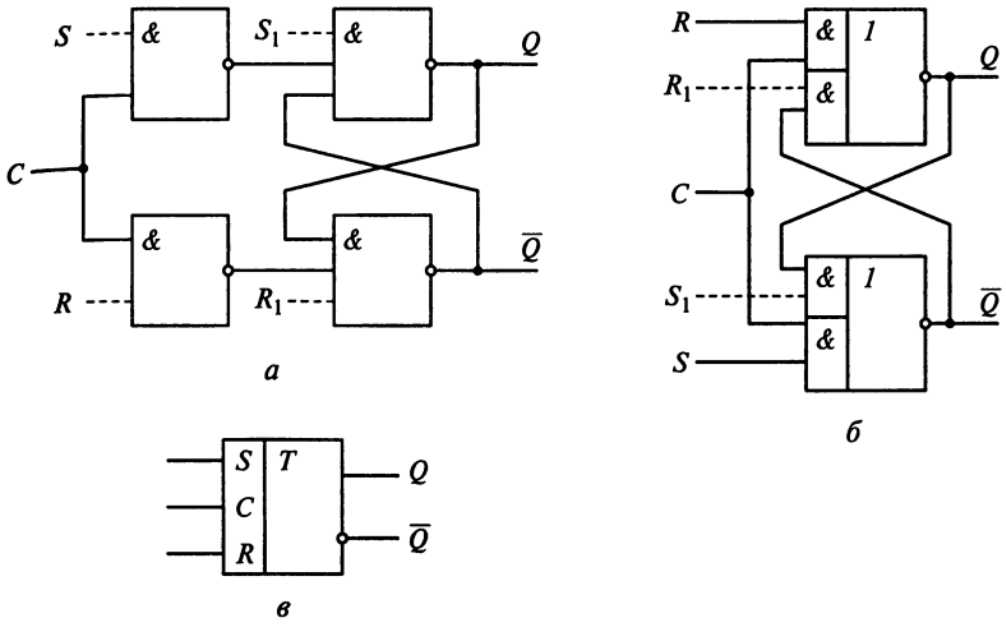


Рисунок 3.9 – Синхронный RS-триггер
а – на элементах И-НЕ, *б* – на элементах И-ИЛИ-НЕ, *в* – УГО

Асинхронный и синхронный *D*-триггеры. В вычислительной технике широко применяют *D*-триггеры, которые реализуют функцию временно́й задержки входного сигнала (*D* - англ. delay, задержка). Также *D*-триггеры имеют один информационный вход. Логика работы асинхронного *D*-триггера описывается таблицей переходов (таблица 3.4*а*).

Таблица 3.4 – Переходы *D*-триггера (*а* – асинхронного, *б* – синхронного)

(*а*)

Вход	Состояния	
D	0	1
0	0	0
1	1	1

(*б*)

Входы		Состояния	
D	*C*	0	1
0	0	0	1
0	1	0	0
1	0	0	1
1	1	1	1

В асинхронном *D*-триггере состояние (выходной сигнал) Q_{t+1} повторяет значение входного сигнала D_t поэтому асинхронный *D*-триггер по существу не является элементом памяти и рассматривается только как основа для построения синхронного *D*-триггера.

Функциональная схема и УГО синхронного *D*-триггера, построенного на основе синхронного *RS*-триггера, показаны на рисунке 3.10. Для преобразования *RS*-триггера в *D*-триггер сигнал *D* подается на вход *S* непосредственно, а на вход *R* — через инвертор. Если при *C* = 1 на вход *D* подать сигнал «1», то триггер перейдет в состояние «1», а при подаче сигнала *D* = 0 в триггер будет записан «0». Таким образом, для записи в *D*-триггер единицы на вход *D* нужно подать сигнал «1», а для записи нуля — сигнал «0» (так как триггер синхронный, на вход *C* необходимо в обоих случаях подавать сигнал «1»). Это делает *D*-триггер удобным для использования в схемах статической памяти, так как для записи достаточно иметь одну линию на разряд данных. При этом сигнал *C* является общим для всех разрядов записываемых данных.

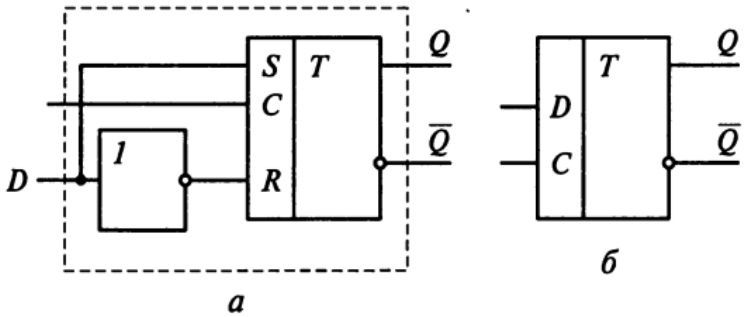


Рисунок 3.10 – Синхронный *D*-триггер (*а* – схема; *б* – УГО)

Логику работы синхронного *D*-триггера описывает таблица 3.4*б*. Эту логику можно охарактеризовать выражением «что надо записать в *D*-триггер, то и подается на его вход».

Наличие входа синхронизации позволяет записывать новые данные в триггер только в определенные моменты времени (при *C*= 1). В промежутках между ними данные в триггере сохраняются без изменения. При чтении данных из триггера его состояние также не меняется.

***T*-триггер**. Этот триггер имеет один информационный вход. Логику работы асинхронного *T*-триггера характеризует таблица переходов (таблица 3.5*а*).

Таблица 3.5 – Переходы *T*-триггера (*а* – асинхронного, *б* – синхронного)

(*а*)

Входы	Состояния	
T	0	1
0	0	1
1	1	0

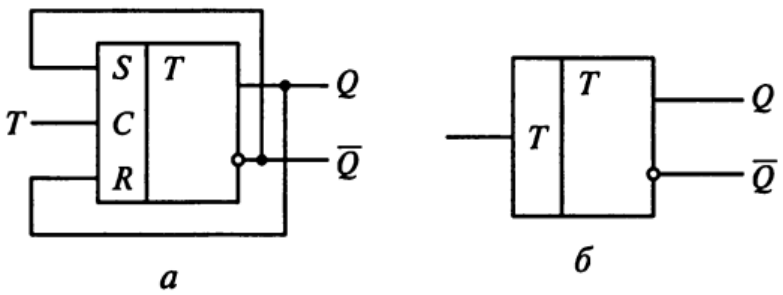
(*б*)

Входы		Состояния	
C	*T*	0	1
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

При *T* = 1 асинхронный *T*-триггер меняет свое состояние на противоположное, а при *T* = 0 состояние триггера не изменяется. (Аналогичную логику работы имеет кнопочный выключатель настольной лампы, который меняет состояние лампы при каждом нажатии кнопки.)

Так как *T*-триггер суммирует (или подсчитывает) по модулю два числа единиц, поступающих на его информационный вход, то *T*-триггер называют также **триггером со счетным входом.**

Логику работы синхронного *T*-триггера описывает таблица 3.5*б*.

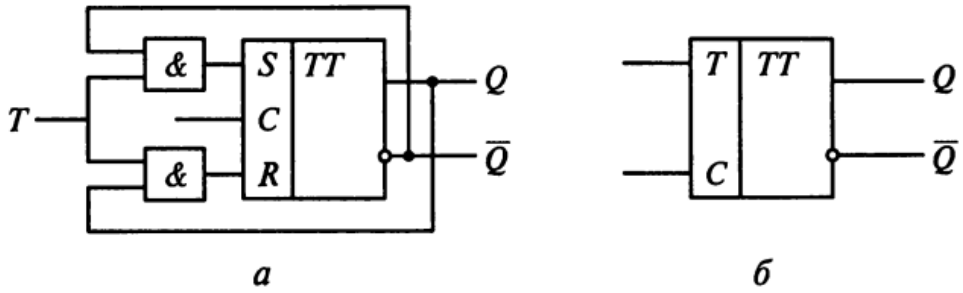


а *б*

Рисунок 3.11 – Асинхронный *T*-триггер (*а* – схема; *б* – УГО)

При *C* = 0 триггер не изменяет своего состояния, а при *C* = 1 работает как асинхронный *T*-триггер. Функциональная схема Г-триггера может быть

построена на основе синхронного *RS*-триггера (однотактного или двухтактного). Схемы асинхронного и синхронного *T*- триггеров показаны на рисунках 3.11 и 3.12 соответственно.



а *б*

Рисунок 3.12 – Синхронный *T*-триггер (*а* – схема; *б* – УГО)

Поскольку на этих схемах сигнал с выхода триггера поступает на его же вход, триггер должен во время переключения сохранять состояние и одновременно воспринять новую информацию. Для устойчивой работы в этом случае целесообразно использовать двухтактные триггеры.

***JK*-триггер**. Такие триггеры называют универсальными. Универсальность схемы *JK*-триггера состоит в том, что простой коммутацией входов и выходов можно получать схемы других типов триггеров.

JK -триггер имеет два информационных входа. Вход *J* используется для установки триггера в состояние «1», а вход *K* — в состояние «0», т. е. входы *J* и *K* аналогичны входам *S* и *R RS*-триггера. Отличие *JK*-триггера от *RS*-триггера заключается в том, что на входы *J* и *K* могут одновременно поступать сигналы «1», в этом случае *JK*-триггер изменяет свое состояние. Таким образом, он работает так же, как *RS*-триггер, за исключением комбинации сигналов *J*=1; *K*=1, при которой он работает как *T*-триггер.

При *C* = 1 переходы *JK*-триггера описывает таблица 3.6.

Таблица 3.6 – Переходы *JK*-триггера

Входы		Состояния	
J	*K*	0	1
0	0	0	1
0	1	0	0
1	0	1	1
1	1	1	0

Функциональная схема двухтактного *JK*-триггера и его УГО показаны на рис. 3.13. Этот триггер представляет собой комбинацию *RS*- и *T*-триггеров, что согласуется с логикой его работы. Примеры построения других типов триггеров на основе *JK*-триггера представлены на рис. 3.14. Следует

отметить, что триггер любого типа можно преобразовать в любой другой триггер.

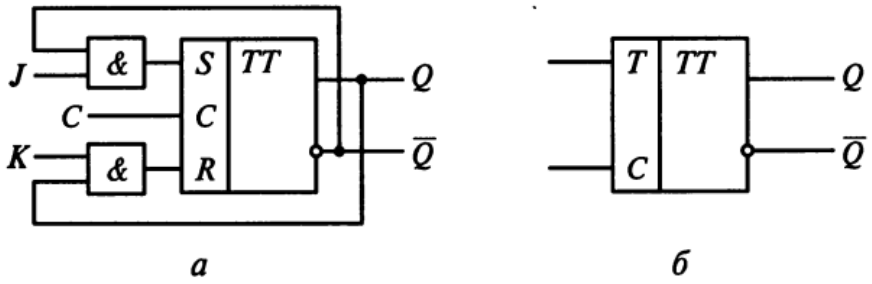


Рисунок 3.13 – Двухтактный *JK*-триггер (*а* – схема; *б* – УГО)

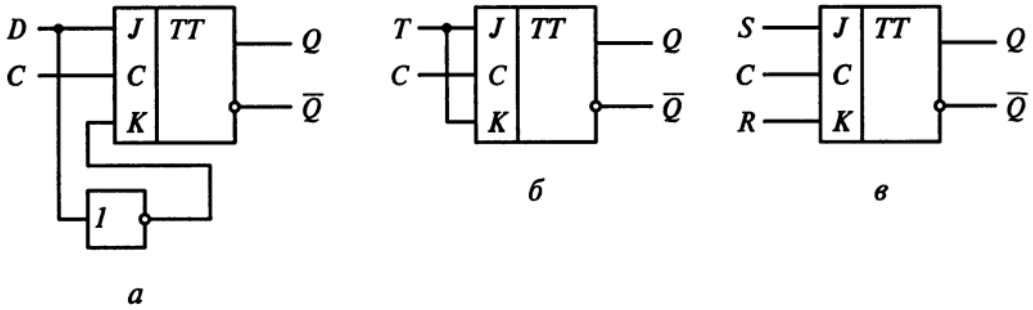


Рисунок 3.14 – Схемы преобразования *JK*-триггера
(*а* – *D*-триггер, *б* – *T*-триггер, *в* – *RS*-триггер)

3.2 Машина Тьюринга

Переработка информации в системах управления и связи, решение различного рода задач на вычислительных машинах или вручную - все эти процессы целенаправленной переработки информации могут рассматриваться как некоторая упорядоченная последовательность операции. *Предписание, определяющее содержание и последовательность операций, переводящих исходные данные в искомый результат, называется алгоритмом.*

Примерами простейших алгоритмов могут служить последовательности операций, позволяющие выполнять арифметические действия, решать алгебраические уравнения, вычислять площади фигур. Способ построения схемы логического автомата на основании заданной логической функции, которую он должен реализовать, можно также рассматривать как *алгоритм* синтеза схемы логического автомата. Как упоминалось ранее, переработка информации в управляющем устройстве для формирования сигналов управления также осуществляется при помощи определенного алгоритма - алгоритма управления.

Любой алгоритм должен удовлетворять требованиям: *определенности*, *массовости* и *результативности*. Под определенностью алгоритма здесь подразумевается его точность и однозначность, не оставляющая места для произвола. Массовость алгоритма означает его применимость для целого класса задач (а не для одной задачи) при различных вариантах исходных

данных. Требованию результативности отвечает алгоритм, приводящий к получению искомого результата после выполнения *конечного числа* операций.

Представим совокупность всех возможных исходных данных, перерабатываемых каким-либо алгоритмом *A* в виде последовательности

$$\alpha_1 , \alpha_2 , \dots , \alpha_i , \dots ,$$

а все возможные результаты - в виде последовательности

$$\beta_1 , \beta_2 , \dots , \beta_j , \dots .$$

Тогда любой алгоритм A_k , преобразующий исходные данные α_i в результат β_j, можно свести к вычислению функции φ_k , указывающей номер результата *j* по номеру совокупности исходных данных *i*

$$j = \varphi_k (i).$$

Но индексы *i* и *j* являются целыми числами и всегда могут быть записаны в двоичной системе счисления при помощи конечного набора нулей и единиц. При этом функция φ_k может рассматриваться как логическая функция, преобразующая ситуацию *i* на входе автомата в ситуацию *j* на его выходе.

Следовательно, любой алгоритм в принципе может быть реализован при помощи соответствующего дискретного автомата.

Английский математик Тьюринг предложил абстрактную схему автомата, принципиально пригодного для реализации любого алгоритма. Этот автомат, получивший название "*Машина Тьюринга*" является *автоматом с бесконечной памятью.*

Запоминающим устройством в машине Тьюринга служит лента, разделенная на ячейки №l, №2, ... так, что эта лента имеет начало (ячейка №1), но не имеет конца (простирается в бесконечность как последовательность натуральных чисел).

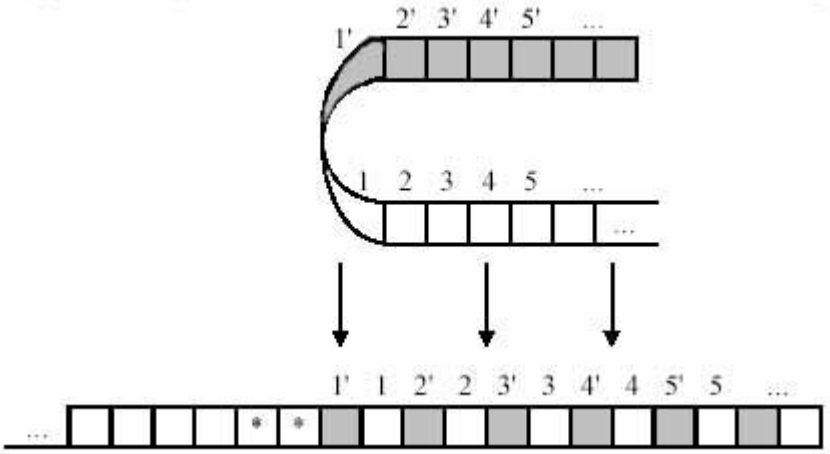


Рисунок 3.15 – Машина Тьюринга

В каждой из ячеек может быть записан нуль или единица. Над лентой перемещается головка T, управляемая автоматом L, обладающим конечной памятью. Автомат L работает тактами. На его вход поступает информация о символе (0 или 1), считываемой головкой с ленты. Под влиянием команд, получаемых каждый такт от автомата L, головка может оставаться на месте или передвигаться на одну ячейку вправо или влево. Одновременно головка получает от автомата L команды, под влиянием которых она может заменить символ, записанный в ячейке, над которой она находится.

Действие машины Тьюринга однозначно определяется исходным заполнением ячеек ленты и оператором преобразования управляющего автомата, который может быть задан таблицей переходов. Обозначим через S_i (S_0=0, S_1=l) символ, считываемый головкой, через R_J (R_0 – стоп, R_1 - влево, R_2 - вправо) - команду на перемещение головки и через q_k (k=1,2, … , n) - состояние управляющего автомата. Тогда его таблица переходов может быть представлена таким образом (таблица 3.7).

Как видно из таблицы 3.7, действие автомата зависит от входа S и его состояния q. Определенным значениям S_i и q_i соответствует определенная совокупность значений трех величин: S, R и q, обозначающих соответственно: какой символ S запишет головка на ленту, какова будет команда R на перемещение головки и в какое новое состояние q перейдет автомат L. Следует иметь в виду, что среди состояний q автомата L должно быть по крайней мере одно такое его состояние q^*, при котором: головка не изменяет символа S, команда R=R_0 (стоп) и автомат остается в состоянии покоя q^*. Придя в состояние q^*, автомат заканчивает выполнение алгоритма, и дальнейшая работа машины Тьюринга прекращается.

Таблица 3.7 – Таблица переходов

Состояние	Вход	
	S_0=*0*	S_1=*1*
q_1	S_0 , R_2 , q_k	S_1 , R_1 , q_m
q_2	S_1 , R_0 , q_s	S_0 , R_1 , q_l
q_3	S_1 , R_1 , q_p	S_0 , R_2 , q_s
…	…	…

Пусть, например, таблица перехода автомата L имеет следующий вид (таблица 3.8)

Таблица 3.8 – Таблица переходов

Q	*S*	
	0	*1*
q^*	*0*, R_0 , q^*	*1*, R_0 , q^*
q_1	*1*, R_0 , q^*	*1*, R_2 , q_1

Тогда, если в начальный момент автомат находится в состоянии q_1, а головка расположена над ячейкой, в которой записан символ *1*, то головка будет перемещаться вправо до тех пор, пока не обнаружит ячейку с символом *0*, заменит его символом *1* и остановится. Если начальное состояние системы (состояние в нулевой такт) и заполнение ленты соответствуют требуемым, то, согласно таблице 3.8 система в последующие два такта перейдет в состояния, показанные в таблице 3.8, и на втором такте остановится.

Приведенный пример показывает лишь одну из самых простых задач, решаемых машиной Тьюринга. При соответствующем расширении таблицы переходов можно заставить машину Тьюринга решать сколь угодно сложные задачи и, во всяком случае, любые задачи, которые способна решать какая-либо другая машина. Однако при этом следует иметь в виду, что практическое применение автоматов, построенных по схеме машины Тьюринга, нецелесообразно, ибо число шагов, необходимых для решения сколько-нибудь сложных задач, чрезвычайно велико, а значит, велико и время решения.

Тем не менее, теория машин Тьюринга имеет большое значение для решения таких важных вопросов, как существование алгоритма решения того или иного класса задач, а значит, могут выполняться автоматически, в частности, таким универсальным автоматам как цифровая вычислительная машина.

3.3 Кодирование информации

Коды как средство тайнописи появились в глубокой древности. Известно, что еще древнегреческий историк Геродот в V в. до н. э. приводил примеры писем, понятных лишь адресату. Секретная азбука использовалась Юлием Цезарем. Над созданием различных секретных шифров работали такие известные ученые средневековья, как Ф. Бэкон, Д. Кардане и др. Появлялись очень хитрые шифры и коды, которые, однако, с течением времени расшифровывались и переставали быть секретом. Первым кодом, предназначенным для передачи сообщений по каналам связи, был код С. Морзе, содержащий разное количество символов для кодирования букв и цифр. Затем появился код Ж. Бодо, используемый в телеграфии, в котором все буквы или цифры содержат одинаковое количество символов. В качестве символов может выступать наличие или отсутствие (пробел) импульса в электрической цепи в данный момент.

Коды, использующие два различных элементарных сигнала, называются *двоичными*. Эти сигналы удобно обозначать символами 0 и 1. Тогда кодовое слово будет состоять из последовательностей нулей и единиц.

Двоичное (бинарное) кодирование тесно связано с принципом дихотомии, который реализуется в графическом методе представления двоичной информации в виде графов.

Ранее были рассмотрены общие и конкретные вопросы кодирования информации в цифровом автомате. Однако эти методы сами по себе еще не

обеспечивают правильность выполнения того или иного алгоритма. Различные алгоритмы выполнения арифметических операций обеспечат правильный результат только в случае, если машина работает без нарушений. При возникновении какого-либо нарушения нормального функционирования результат будет неверным, однако пользователь об этом не узнает, если не будут предусмотрены меры, сигнализирующие о появлении ошибки. Следовательно, с одной стороны, разработчиками машины должны быть предусмотрены меры для создания системы обнаружения возможной ошибки, а с другой стороны, должны быть проработаны меры, позволяющие исправить ошибки. Эти функции следует возложить на систему контроля работы цифрового автомата.

Система контроля - совокупность методов и средств, обеспечивающих определение правильности работы автомата в целом или его отдельных узлов, а также автоматическое исправление ошибки. Ошибки в работе цифрового автомата могут быть вызваны либо выходом из строя какой-либо детали, либо отклонением от нормы параметров, например изменением напряжения питания или воздействием внешних помех. Вызванные этими нарушениями ошибки могут принять постоянный или случайный характер. Постоянные ошибки легче обнаружить и выявить. Случайные ошибки, обусловленные кратковременными изменениями параметров, наиболее опасны, и их труднее обнаружить.

Поэтому система контроля должна строиться с таким расчетом, чтобы она позволяла обнаружить и по возможности исправить любые нарушения. При этом надо различать следующие виды ошибок результата:

1) возникающие из-за погрешностей в исходных данных;

2) обусловленные методическими погрешностями;

3) появляющиеся из-за возникновения неисправностей в работе машины.

Первые два вида ошибок не являются объектом для работы системы контроля. Погрешности перевода или представления числовой информации в разрядной сетке автомата приведут к возникновению погрешности в результате решения задачи. Эту погрешность можно заранее рассчитать и, зная ее максимальную величину, правильно выбрать длину разрядной сетки машины. Методические погрешности также учитываются предварительно.

Проверка правильности функционирования отдельных устройств машины и выявление неисправностей может осуществляться по двум направлениям:

профилактический контроль, задача которого - предупреждение появления возможных ошибок в работе;

оперативный контроль, задача которого - проверка правильности выполнения машиной всех операций.

Решение всех задач контроля становится возможным только при наличии определенной избыточности информации. Избыточность может быть создана либо аппаратными (схемными) средствами, либо логическими или информационными средствами.

К методам логического контроля, например, можно отнести следующие приемы. В ЭВМ первого и второго поколения отсутствие системы оперативного контроля приводило к необходимости осуществления «двойного счета», когда каждая задача решалась дважды и в случае совпадения ответов принималось решение о правильности функционирования ЭВМ.

Если в процессе решения какой-то задачи вычисляются тригонометрические функции, то для контроля можно использовать известные соотношения между этими функциями, например, $\sin^2 \alpha + \cos^2 \alpha = 1$. Если это соотношение выполняется с заданной точностью на каждом шаге вычислений, то можно с уверенностью считать, что ЭВМ работает правильно.

Вычисление определенного интеграла с заданным шагом интегрирования можно контролировать сравнением полученных при этом результатов с теми результатами, которые соответствуют более крупному шагу. Такой «сокращенный» алгоритм даст, видимо, более грубые оценки и по существу требует дополнительных затрат машинного времени.

Все рассмотренные примеры свидетельствуют о том, что такие методы контроля позволяют лишь зафиксировать факт появления ошибки, но не определяют место, где произошла эта ошибка. Для оперативного контроля работы ЭВМ определение места, где произошла ошибка, т. е. решение задачи поиска неисправности, является весьма существенным вопросом.

Основные понятия теории кодирования

Задача кодирования информации представляется как некоторое преобразование числовых данных в заданной системе счисления. В частном случае эта операция может быть сведена к группированию символов (представление в виде триад или тетрад) или представлению в виде символов (цифр) позиционной системы счисления. Так как любая позиционная система не несет в себе избыточности информации и все кодовые комбинации являются разрешенными, использовать такие системы для контроля не представляется возможным.

Систематический код - код, содержащий в себе кроме информационных контрольные разряды.

В контрольные разряды записывается некоторая информация об исходном числе. Поэтому можно говорить, что систематический код обладает *избыточностью*. При этом абсолютная избыточность будет выражаться количеством контрольных разрядов k, а относительная избыточность - отношением k/n , где $n = m + k$ - общее количество разрядов в кодовом слове (m - количество информационных разрядов).

Понятие корректирующей способности кода обычно связывают с возможностью обнаружения и исправления ошибки. Количественно корректирующая способность кода определяется вероятностью обнаружения или исправления ошибки. Если имеем n-разрядный код и вероятность

искажения одного символа p, то вероятность того, что искажены k символов, а остальные n - k символов не искажены, по теореме умножения вероятностей будет

$$w = p^k(1-p)^{n-k}.$$

Число кодовых комбинаций, каждая из которых содержит k искаженных элементов, равна числу сочетаний из n по k:

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

Тогда полная вероятность искажения информации

$$P_\Sigma = \sum_{i=1}^{k} \frac{n!}{i!(n-i)!} p^i (1-p)^{n-i}.$$

Так как на практике $p = 10^{-3} \ldots 10^{-4}$, наибольший вес в сумме вероятностей имеет вероятность искажения одного символа. Следовательно, основное внимание нужно обратить на обнаружение и исправление одиночной ошибки.

Корректирующая способность кода связана также с понятием кодового расстояния.

Кодовое расстояние $d(A, B)$ для кодовых комбинаций A и B определяется как вес третьей кодовой комбинации, которая получается поразрядным сложением исходных комбинаций по модулю 2 (операция "исключающее ИЛИ", XOR), а говоря проще - количество различающихся разрядов в двух кодовых комбинациях.

Вес кодовой комбинации $V(A)$ - количество единиц, содержащихся в кодовой комбинации.

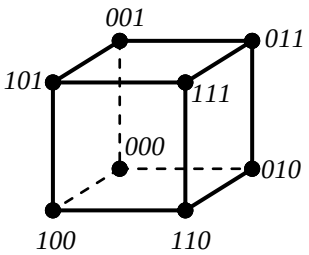


Рисунок 3.16 – Геометрическое представление кодов

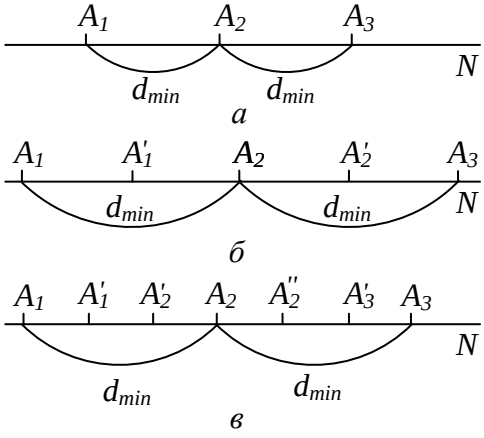


Рисунок 3.17 – Кодовые расстояния

Коды можно рассматривать и как некоторые геометрические (пространственные) фигуры. Например, триаду можно представить в виде единичного куба, имеющего координаты вершин, которые отвечают двоичным символам (рисунок 3.16). В этом случае кодовое расстояние воспринимается как сумма длин ребер между соответствующими вершинами

куба (принято, что длина одного ребра равна 1). Оказывается, что любая позиционная система отличается тем свойством, что минимальное кодовое расстояние равно 1 (рисунок 3.17*а*).

В теории кодирования показано, что систематический код способен обнаружить ошибки только тогда, когда минимальное кодовое расстояние для него больше или равно 2*t*, т.е.

$$d_{\min} \geq 2t, \qquad (3.6)$$

где *t* - кратность обнаруживаемых ошибок (в случае одиночных ошибок *t* = 1 и т. д.).

Это означает, что между соседними разрешенными кодовыми словами должно существовать по крайней мере одно кодовое слово (рисунок 3.17*б*,*в*).

В тех случаях, когда необходимо не только обнаружить ошибку, но и исправить ее (т. е. указать место ошибки), минимальное кодовое расстояние должно быть

$$d_{\min} \geq 2t+1. \qquad (3.7)$$

Допустим, имеем следующий набор кодовых комбинаций:

000
001
010
011
100
101
110
111

Геометрическая модель этого кода – куб, представленный на рисунке 3.16

Для рассматриваемого кода $d_{min}= 1$. Учитывая, что рассматриваемый код по построению является неизбыточным, можно утверждать, что любой неизбыточный код имеет $d_{min}= 1$ и наоборот, если $d_{min}= 1$, код является неизбыточным.

Из геометрической модели кода видно, что любая, даже одиночная, ошибка не может быть обнаружена кодом с $d_{min}= 1$, так как любая кодовая комбинация, в которой произошла ошибка, переместившись на один кодовый переход, совпадет с соседней кодовой комбинацией и будет разрешенной.

Для построения простейшего избыточного кода, который может обнаруживать одну ошибку, нужно отобрать рабочие комбинации на расстоянии $d(A, B) \geq 2$.

В рассматриваемом коде можно выбрать следующие комбинации:

$$\left.\begin{matrix}010\\100\\111\\001\end{matrix}\right| \rightarrow \mathrm{M_p} = 4$$

где M_p – число разрешенных (или рабочих) комбинаций.

Избыточность полученного кода

$$R = \frac{k}{n} = \frac{n-m}{n} = \frac{1}{3} \approx 33\%.$$

Если требуется обнаруживать две ошибки, то рабочих комбинаций будет только две (например, 000 и 111), минимальное кодовое расстояние в этом случае $d_{min} = 3$, избыточность

$$R = \frac{k}{n} = \frac{n-m}{n} = \frac{2}{3} \approx 67\%.$$

Если требуется обнаруживать три ошибки, $d_{min} \geq 4$, что невозможно обеспечить в рассматриваемом коде, так как кодовое расстояние $d(A, B) \leq 3$.

Если бы сообщения кодировались неизбыточным кодом, то для получения четырех кодовых комбинаций потребовалось бы $m = 2$ информационных элементов.

Далее для иллюстрации неравенства о возможности исправления ошибки (3.7) приведем следующий пример.

Возьмем три рядом стоящих слова (термин «рядом стоящие» означает, что между этими словами других слов быть не может):

$$a = 0100, \quad b = 0110, \quad c = 0010.$$

Далее получим соответствующие им кодовые слова

$$f_a = 1110100, \quad f_b = 0010110, \quad f_c = 1100010.$$

Кодовые расстояния между этими словами равны:

$$d(f_a, f_b) = 3,\ d(f_b, f_c) = 4,\ d(f_a, f_c) = 3.$$

Предположим, что при передачи информации в канале связи произошел сбой и второе кодовое слово изменилось на $f_b' = 0110110$. Тогда по минимальному кодовому расстоянию мы все же сможем определить, что из трех возможных слов передавалось скорее всего слово f_b, так как:

$$d(f_a, f_b') = 2, \quad d(f_b, f_b') = 1, \quad d(f_c, f_b') = 3.$$

Если же были допущены две ошибки, например, в пятом и четвертом разрядах, то становится непонятным, какое из кодовых слов – f_b или f_c – на самом деле передавалось, так как $f_b'' = 0111110$,

$$d(f_a, f_b'') = 3, \quad d(f_b, f_b'') = 2, \quad d(f_c, f_b'') = 2.$$

Отсюда, чем больше минимальное кодовое расстояние между словами, тем лучше защищен код от помех. По вышеприведенному неравенству можно установить, что при $d_{min} = 3$ число обнаруженных и исправленных ошибок не может быть больше одной ($m \leq 1$), поскольку $m \leq (d_{min} - 1)/2$.

Существуют коды, в которых невозможно выделить абсолютную избыточность. Неявная избыточность характерна также для кодов типа «*k* из *n*». Примером является код «2 из 5», который часто используется для представления информации. Суть его в том, что в слове из пяти разрядов только два разряда имеют единичное значение.

Методы эффективного кодирования информации

Информационную избыточность можно ввести разными путями. Рассмотрим один из путей эффективного кодирования.

В ряде случаев буквы сообщений преобразуются в последовательности двоичных символов. Учитывая статистические свойства источника сообщения, можно минимизировать среднее число двоичных символов, требующихся для выражения одной буквы сообщения, что при отсутствии шума позволит уменьшить время передачи.

Такое эффективное кодирование базируется на основной теореме Шеннона для каналов без шума, в которой доказано, что сообщения, составленные из букв некоторого алфавита, можно закодировать так, что среднее число двоичных символов на букву будет сколь угодно близко к энтропии источника этих сообщений, но не меньше этой величины.

Теорема не указывает конкретного способа кодирования, но из нее следует, что при выборе каждого символа кодовой комбинации необходимо стараться, чтобы он нес максимальную информацию. Следовательно, каждый символ должен принимать значения 0 и 1 по возможности с равными вероятностями и каждый выбор должен быть независим от значений предыдущих символов.

При отсутствии статистической взаимосвязи между буквами конструктивные методы построения эффективных кодов были даны впервые К.Шенноном и Н. Фано. Их методики существенно не различаются, поэтому соответствующий код получил название кода Шеннона-Фано.

Код строится следующим образом: буквы алфавита сообщений выписываются в таблицу в порядке убывания вероятностей. Затем они разделяются на две группы так, чтобы суммы вероятностей в каждой из групп были по возможности одинаковы. Всем буквам верхней половины в качестве первого символа приписывается 1, а всем нижним - 0. Каждая из полученных групп, в свою очередь, разбивается на две подгруппы с одинаковыми суммарными вероятностями и т. д. Процесс повторяется до тех пор, пока в каждой подгруппе останется по одной букве.

Рассмотрим алфавит из восьми букв (таблица 3.9). Ясно, что при обычном (не учитывающем статистических характеристик) кодировании для представления каждой буквы требуется три символа.

Таблица 3.9 – Кодирование по методике Шеннона-Фано

Буквы	Вероятности	Кодовые комбинации
z_1	0,22	11
z_2	0,20	101
z_3	0,16	100
z_4	0,16	01
z_5	0,10	001
z_6	0,10	0001
z_7	0,04	00001
z_8	0,02	00000

Вычислим энтропию набора букв:

$$(z) = -\sum_{i=1}^{8} p(z_i)\log p(z_i) \approx 2{,}76$$

и среднее число символов на букву

$$l_{ср} = -\sum_{i=1}^{8} p(z_i)\, n(z_i) \approx 2{,}84,$$

где $n(z_i)$ - число символов в кодовой комбинации, соответствующей букве z_i.

Значения z_i и $l_{ср}$ не очень различаются по величине.

Рассмотренная методика Шеннона-Фано не всегда приводит к однозначному построению кода. Ведь при разбиении на подгруппы можно сделать большей по вероятности как верхнюю, так и нижнюю подгруппу.

Множество вероятностей в предыдущей таблице можно было разбить иначе (таблица 3.10).

При этом среднее число символов на букву оказывается равным 2,80. Таким образом, построенный код может оказаться не самым лучшим. При построении эффективных кодов с основанием $q > 2$ неопределенность становится еще больше.

Таблица 3.10 – Кодирование по методике Шеннона-Фано (продолжение)

Буквы	Вероятности	Кодовые комбинации
z_1	0,22	11
z_2	0,20	10
z_3	0,16	011
z_4	0,16	010
z_5	0,10	001
z_6	0,10	0001
z_7	0,04	00001
z_8	0,02	00000

От указанного недостатка свободна методика Д. Хаффмена. Она гарантирует однозначное построение кода с наименьшим для данного распределения вероятностей средним числом символов на букву.

Буквы	*Вероятности*	*Вспомогательные столбцы*						
		1	2	3	4	5	6	7
z_1	0,22	0,22	0,22	0,26	0,32	0,42	0,58	1
z_2	0,20	0,20	0,20	0,22	0,26	0,32	0,42	
z_3	0,16	0,16	0,16	0,20	0,22	0,26		
z_4	0,16	0,16	0,16	0,16	0,20			
z_5	0,10	0,10	0,16	0,16				
z_6	0,10	0,10	0,10					
z_7	0,04	0,06						
z_8	0,02							

Рисунок 3.18 – Кодирование по методике Хаффмена

Для двоичного кода методика сводится к следующему. Буквы алфавита сообщений выписываются в основной столбец в порядке убывания вероятностей. Две последние буквы объединяются в одну вспомогательную букву, которой приписывается суммарная вероятность. Вероятности букв, не участвовавших в объединении, и полученная суммарная вероятность снова располагаются в порядке убывания вероятностей в дополнительном столбце, а две последние объединяются. Процесс продолжается до тех пор, пока не получим единственную вспомогательную букву с вероятностью, равной единице.

Для составления кодовой комбинации, соответствующей данному сообщению, необходимо проследить путь перехода сообщений по строкам и столбцам таблицы. Для наглядности строится кодовое дерево. Из точки, соответствующей вероятности 1, направляются две ветви, причем ветви с большей вероятностью присваивается символ 1, а с меньшей - 0. Такое последовательное ветвление продолжаем до тех пор, пока не дойдем до каждой буквы (рисунок 3.19).

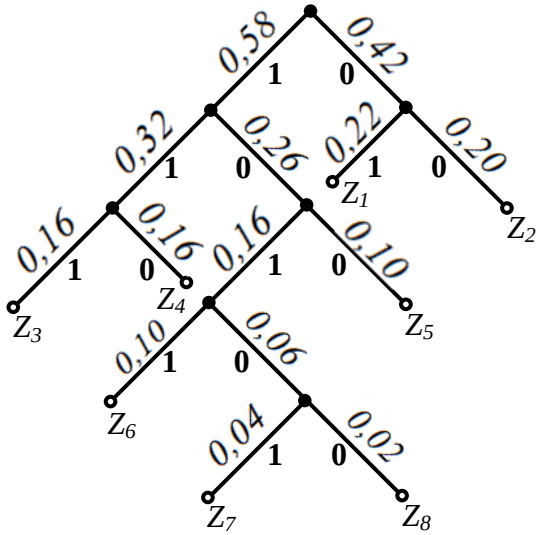


Рисунок 3.19 – Кодовое дерево

Теперь, двигаясь по кодовому дереву сверху вниз, можно записать для каждой буквы соответствующую ей кодовую комбинацию:

Таблица 3.11 – Кодирование по методике Хаффмена

z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
01	00	111	110	100	1011	10101	10100

Кодирование по методу четности-нечетности

Если в математическом коде выделен один контрольный разряд (k = 1), то к каждому двоичному числу добавляется один избыточный разряд и в него записывается 1 или 0 с таким условием, чтобы сумма цифр в каждом числе была по модулю 2 равна 0 для случая четности или 1 для случая нечетности.

Появление ошибки в кодировании обнаружится по нарушению четности (нечетности). При таком кодировании допускается, что может возникнуть только одна ошибка. В самом деле, для случая четности правильной будет только половина возможных комбинаций. Чтобы одна допустимая комбинация превратилась в другую, должно возникнуть по крайней мере два нарушения или четное число нарушений. Пример реализации метода четности представлен в таблице 3.12.

Таблица 3.12 – Пример реализации метода четности

Число	Контрольный разряд	Проверка
10101011	1	0
11001010	0	0
10010001	1	0
11001011	0	1-нарушение

Такое кодирование имеет минимальное кодовое расстояние, равное 2.

Можно представить и несколько видоизмененный способ контроля по методу четности - нечетности. Длинное число разбивается на группы, каждая из которых содержит l разрядов. Контрольные разряды выделяются всем группам по строкам и по столбцам согласно следующей схеме:

a_1	a_2	a_3	a_4	a_5	k_1
a_6	a_7	a_8	a_9	a_{10}	k_2
a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	k_3
a_{16}	a_{17}	a_{18}	a_{19}	a_{20}	k_4
a_{21}	a_{22}	a_{23}	a_{244}	a_{25}	k_5
k_6	k_7	k_8	k_9	k_{10}	

Рисунок 3.20 – Выделение контрольных разрядов по строкам и столбцам

Увеличение избыточности информации приводит к тому, что появляется возможность не только обнаружить ошибку, но и исправить ее. Пусть произошла неисправность в каком-то из разрядов этого числа (представим, что разряд a_{18} изменил состояние, т. е. $a_{18} = 1$). Это приведет к тому, что при проверке на четность сумма $\Sigma_i(a_i+k_i)$ по соответствующим строкам и столбцам изменится для значений, которые содержат элемент a_{18}, т. е. это будет четвертая сверху строка и третий слева столбец. Следовательно, нарушение четности по этой строке и столбцу означает обнаружение не только самой ошибки, но и места, где возникла ошибка. Изменив содержимое отмеченного разряда (в данном случае a_{18}) на противоположное, можно исправить ошибку.

Коды Хэмминга

Коды, предложенные американским ученым Р. Хэммингом, способны не только обнаружить, но и исправить одиночные ошибки. Эти коды - систематические.

Предположим, что имеется код, содержащий m информационных разрядов и k контрольных разрядов. Запись на k позиций определяется при проверке на четность каждой из проверяемых k групп информационных символов. Пусть было проведено k проверок. Если результат проверки свидетельствует об отсутствии ошибки, то запишем 0, если есть ошибка, то запишем 1 . Запись полученной последовательности символов образует двоичное, контрольное число, указывающее номер позиции, где произошла ошибка. При отсутствии ошибки в данной позиции последовательность будет содержать только нули. Полученное таким образом число описывает $n = (m + k + 1)$ событий. Следовательно, справедливо неравенство

$$2^k \geq (m + k + 1).$$

Определить максимальное значение m для данного k можно из следующего:

n… 1 2 3 4… 8…15 16…31 32…63 64
m…0 0 1 1… 4…11 11…26 26…57 57
k… 1 2 2 3… 4…4 5…5 6…6 7

Определим теперь позиции, которые надлежит проверить в каждой из k проверок. Если в кодовой комбинации ошибок нет, то контрольное число содержит только нули. Если в первом разряде контрольного числа стоит 1, то, значит, в результате первой проверки обнаружена ошибка. Имея таблицу двоичных эквивалентов для десятичных чисел, можно сказать, что, например, первая проверка охватывает позиции 1, 3, 5, 7, 9 и т. д., вторая проверка — позиции 2, 3, 6, 7, 10.

Проверка	**Проверяемые разряды**
1...	1,3,5,7,9,11,13,15...
2...	2,3,6,7, 10, 11, 14, 15, 18, 19,22,23...
3...	4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23...
4...	8,9,10,11,12,13,14,15,24...

Теперь нужно решить, какие из позиций целесообразнее применить для передачи информации, а какие — для ее контроля. Преимущество использования позиций 1, 2, 4, 8, ... для контроля в том, что данные позиции встречаются только в одной проверяемой группе символов.

В таблице 3.13 представлены примеры кодирования информации по методу Хэмминга для семиразрядного кода. Как видно из таблицы 3.13, в этом случае n=7, m=4, k=3 и контрольными будут разряды 1, 2, 4.

Таблица 3.13 – Кодирование информации по методу Хэмминга для семиразрядного кода

Разряды двоичного кода							Кодируемая десятичная информация
1 k_1	2 k_2	3 m_1	4 k_3	5 m_2	6 m_3	7 m_4	
0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1
0	1	0	1	0	1	0	2
1	0	0	0	0	1	1	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	1	5
1	1	0	0	1	1	0	6
0	0	0	1	1	1	1	7
1	1	1	0	0	0	0	8
0	0	1	1	0	0	1	9
1	0	1	1	0	1	0	10
0	1	1	0	0	1	1	11
0	1	1	1	1	0	0	12
1	0	1	0	1	0	1	13
0	0	1	0	1	1	0	14
1	1	1	1	1	1	1	15

По методу Хэмминга могут быть построены коды разной длины, этом чем больше длина кода, тем меньше относительная избыточность. Например, для контроля числа, имеющего 48 двоичных разрядов, потребуется только шесть дополнительных (избыточных) разрядов. Коды Хэмминга используют в основном для контроля передачи информации по каналам связи, что имеет место в вычислительных системах с телеобработкой данных или в системах коллективного пользования.

3.4 Системы счисления

Системой счисления называется совокупность приемов наименования и записи чисел.

В любой системе счисления для представления чисел выбираются некоторые символы (слова или знаки), называемые *базисными числами*, а все остальные числа получаются в результате каких-либо операций из базисных чисел данной системы исчисления. Символы, используемые для записи чисел, могут быть любыми, только они должны быть разными и значение каждого из них должно быть известно. В современном мире наиболее распространенным является представление чисел посредством арабских цифр 0, 1,2, 3, 4, 5, 6, 7, 8, 9 - специальных знаков, используемых для записи чисел. Системы счисления различаются выбором базисных чисел и правилами образования из них остальных чисел. Например, в римской системе счисления базисными являются числа 1, 5, 10, 50, 100, 500, 1000, которые обозначаются знаками I, V, X, L, C, D, M, а другие получаются путем сложения и вычитания базисных: если цифра справа меньше или равна цифре слева, то эти цифры складываются; если цифра слева меньше, чем цифра справа, то левая цифра вычитается из правой. Так, например, число 146 в римской системе счисления имеет вид CXLVI (C-100, XL-40, VI-6),

здесь «сорок» получается посредством вычитания из «пятидесяти» числа «десять», «шесть» - посредством сложения «пяти» и «единицы». Системы счисления, в которых любое число получается путем сложения или вычитания базисных чисел, называются *аддитивными*. При таком представлении чисел правила сложения для небольших чисел очевидны и просты, однако если возникает необходимость выполнять операции сложения над большими числами или операции умножения и деления, то «римская» система счисления оказывается неудобной. В этой ситуации преимущественнее оказываются позиционные системы счисления. Хотя в них, как правило, представления чисел далеко не так просты и очевидны, как в «римской» системе счисления, систематичность представления, основанная на «позиционном весе» цифр, обеспечивает простоту выполнения операций умножения и деления.

В «римской» системе счисления каждый числовой знак в записи любого числа имеет одно и то же значение, т.е. значение числового знака не зависит от его расположения в записи числа. Таким образом, «римская» система счисления не является позиционной системой счисления.

Для изображения (или представления) чисел в настоящее время используются в основном позиционные системы счисления. Привычной для всех является десятичная система счисления. В этой системе для записи любых чисел используется только десять разных знаков (цифр): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Эти цифры введены для обозначения первых десяти последовательных чисел, а следующее число 10 и т.д. обозначается уже без использования новых цифр. Однако введением этого обозначения сделан важный шаг в построении системы счисления: значение каждой цифры поставлено в зависимость от того места, где она стоит в изображении числа.

Таким образом, система называется *позиционной*, если значение каждой цифры (ее вес) изменяется в зависимости от ее положения (позиции) в последовательности цифр, изображающих число.

Десятичная позиционная система счисления основана на том, что десять единиц каждого разряда объединяются в одну единицу соседнего старшего разряда. Таким образом, каждый разряд имеет вес, равный степени 10. Например, в записи числа 343,32 цифра 3 повторена три раза, при этом самая левая цифра 3 означает количество сотен (ее вес равен 10^2); цифра 3, стоящая перед точкой, означает количество единиц (ее вес равен 10^0), а самая правая цифра 3 - количество десятых долей единицы (ее вес равен 10^{-1}), так что последовательность цифр 343,32 представляет собой сокращенную запись выражения:

$$3\times10^2+4\times10^1+3\times10^0+3\times10^{-1}+2\times10^{-2}.$$

Десятичная запись любого числа X в виде последовательности цифр:

$$a_n a_{n-1}\ldots a_1 a_0 a_{-1}\ldots a_{-m}\ldots \qquad (3.8)$$

основана на представлении этого числа в виде полинома:

$$X = a_n\times10^n + a_{n-1}\times10^{n-1}+\ldots+ a_1\times10^1+ a_0\times10^0 + a_{-1}\times10^{-1}+\ldots+ a_{-m}\times10^{-m}\ldots, \qquad (3.9)$$

где каждый коэффициент a_i может быть одним из чисел, для обозначения

которых введены специальные знаки. Запись числа X в виде (3.9) представляет собой просто перечисление всех коэффициентов этого полинома. Точка или запятая, отделяющая целую часть числа от дробной, служит для фиксации конкретных значений каждой позиции в этой последовательности цифр и является началом отсчета.

Число K единиц какого-либо разряда, объединяемых в единицу более старшего разряда, называют *основанием позиционной системы счисления*, а сама система счисления называется *K-ичной*. Например, основанием десятичной системы счисления является число 10; двоичной - число 2; троичной - число 3 и т.д. Для записи произвольного числа в K-ичной системе счисления достаточно иметь K разных цифр a_i, i=1,...,K. Эти цифры служат для обозначения некоторых различных целых чисел, называемых базисными.

Числа можно записать как суммы степеней не только числа 10, но и любого другого натурального числа, большего 1. Например, в Древнем Вавилоне использовалась система счисления с основанием 60. Деление часа на 60 минут, а минуты на 60 секунд заимствовано именно из этой системы счисления. А то, что человечество выбрало в качестве основания системы счисления число 10, вероятно, связано с тем, что природа наделила людей десятью пальцами.

Запись произвольного числа X в K-ичной позиционной системе счисления основывается на представлении этого числа в виде полинома:

$$K=a_nK^n + a_{n-1}K^{n-1} +\ldots+ a_1K^1 + a_0K^0 + a_{-1}K^{-1} +\ldots+ a_{-m}K^{-m} +\ldots, \qquad (3.10)$$

где каждый коэффициент a может быть одним из базисных чисел и изображается одной цифрой. Как и в десятичной системе счисления, число X, представленное в K-ичной системе счисления, можно кратко записать в виде (3.8) путем перечисления всех коэффициентов полинома (3.10) с указанием позиционной точки. Последовательность цифр, стоящая в (3.8), является изображением числа X в K-ичной системе счисления. Базисные числа должны быть выбраны так, чтобы любое число X могло быть представлено в виде полинома (3.10). Обычно в качестве базисных чисел берутся последовательные целые числа от 0 до K-1 включительно.

Все известные позиционные системы счисления являются *аддитивно-мультипликативными*. Особенно отчетливо аддитивно-мультипликативный способ образования чисел из базисных выражен в числительных русского языка, например пятьсот шестьдесят восемь (т.е. пять сотен плюс шесть десятков плюс восемь).

Арифметические действия над числами в любой позиционной системе счисления производятся по тем же правилам, что и в десятичной системе, так как все они основываются на правилах выполнения действий над соответствующими полиномами. При этом нужно только пользоваться теми таблицами сложения и умножения, которые имеют место при данном основании K системы счисления.

Отметим, что во всех позиционных системах счисления с любым основанием K умножения на числа вида K^m , где m - целое число, сводится просто к перенесению запятой у множимого на m разрядов вправо или влево

(в зависимости от знака m), так же как и в десятичной системе.

Для указания того, в какой системе счисления записано число, условимся при его изображении основание системы счисления указывать в виде нижнего индекса при нем, например, 35.64_8.

В современной вычислительной технике в устройствах автоматики и связи широко используется двоичная система счисления. Это система счисления с наименьшим возможным основанием. В ней для изображения числа используются только две цифры: 0 и 1.

Произвольное число X в двоичной системе представляется в виде полинома:

$$X = a_n \times 2^n + a_{n-1} \times 2^{n-1} + \ldots + a_1 \times 2^1 + a_0 \times 2^0 + a_{-1} \times 2^{-1} + \ldots + a_{-m} \times 2^{-m} \ldots, \quad (3.11)$$

где каждый коэффициент a_i может быть либо 0, либо 1.

Так как в двоичной системе счисления для изображения любых чисел используются только две различные цифры, то при построении ЭВМ можно использовать элементы, которые могут находиться только в двух состояниях (например, высокое или низкое напряжение в цепи, наличие или отсутствие электрического импульса и т.п.). Это обстоятельство, а также простота выполнения арифметических операций являются причиной того, что большинство современных ЭВМ используют двоичную систему счисления.

Неудобство использования двоичной системы счисления заключается в громоздкости записи чисел. Это неудобство не имеет существенного значения для ЭВМ. Однако если возникает необходимость кодировать информацию «вручную», например, при составлении программы на машинном языке, то предпочтительнее оказывается пользоваться восьмеричной или шестнадцатеричной системой счисления (в силу их свойств, которые будут отмечены позднее).

В восьмеричной системе счисления базисными числами являются 0, 1, 2, 3, 4, 5, 6, 7. Запись любого числа в этой системе основывается на его разложении по степеням числа восемь с коэффициентами, являющимися указанными выше базисными числами.

Например, десятичное число 83.5 в восьмеричной системе будет изображаться в виде 123.4_8. Действительно, эта запись по определению означает представление числа в виде полинома:

$$1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 + 4 \times 8^{-1} = 64 + 16 + 3 + 4/8 = 83.5.$$

В шестнадцатеричной системе счисления базисными являются числа от нуля до пятнадцати. Эта система отличается от рассмотренных ранее тем, что в ней общепринятых (арабских) цифр не хватает для обозначения всех базисных чисел, поэтому приходится вводить в употребление новые символы. Обычно для обозначения первых десяти целых чисел от нуля до девяти используют арабские цифры, а для следующих целых чисел от десяти до пятнадцати используются буквенные обозначения A, B, C, D, E, F.

Например, десятичное число 175.5 в шестнадцатеричной системе будет записываться в виде AF.8.

Смешанные системы счисления

В ряде случаев числа, заданные в системе счисления с основанием *P*, приходится изображать с помощью цифр другой системы счисления с основанием *Q*, где *Q*<*P*. Такая ситуация возникает, например, когда в ЭВМ, способной непосредственно воспринимать только двоичные числа, необходимо изобразить десятичные числа, с которыми мы привыкли работать. В этих случаях используются смешанные системы счисления, в которых каждый коэффициент *P*-ичного разложения числа записывается в *Q*-ичной системе. В такой системе *P* называется *старшим основанием*, а *Q* - *младшим основанием*, а сама смешанная система называется (*Q* - *P*)-ичной. Для того чтобы запись числа в смешанной системе счисления была однозначной, для представления любой *P*-ичной цифры отводится одно и то же количество *Q*-ичных разрядов, достаточное для представления любого базисного числа *P*-ичной системы. Так, в смешанной двоично-десятичной системе счисления для изображения каждой десятичной цифры отводится четыре двоичных разряда. Например, десятичное число x = 925 в двоично-десятичной системе запишется в виде 1001 0010 0101. Здесь последовательные четверки (*тетрады*) двоичных разрядов изображают цифры 9, 2, 5 записи числа в десятичной системе счисления. Следует обратить внимание, что хотя в двоично-десятичной записи числа и используются только цифры 0 и 1, эта запись отличается от двоичного изображения данного числа. Например, приведенный выше двоичный код в двоичной системе счисления изображает число 2341, а не число 925.

Условимся изображать принадлежность числа к (*Q* - *P*)-ичной системе счисления с помощью нижнего индекса (*Q* - *P*) при данном числе, например: $925_{10}= 100100100101_{2\text{-}10}$

Аналогично рассмотренной выше двоично-десятичной системе можно использовать и другие смешанные системы при различных значениях *P* и *Q*. Особого внимания заслуживает случай, когда $P = Q^z$, где *z* - целое положительное число. В этом случае запись какого-либо числа в смешанной системе тождественно совпадает с изображением этого числа в системе счисления с основанием *Q* (что не имеет места в двоично-десятичной системе в общем случае).

Докажем это утверждение. Рассмотрим произвольное целое число *N*. В *P*-ичной системе счисления это число будет записано в виде

$$p_n p_{n-1} \ldots p_1 p_0,$$

основанном на представлении

$$N=p_n p^n+p_{n-1}p^{n-1}+\ldots+p_1 p^1+p_0, \qquad (3.12)$$

где p_i, i = 0, 1, ... n являются базисными числами этой системы.

Каждый коэффициент p_i будет записываться в *Q*-ичной системе счисления в виде

$$p_i = q_{i,z-1}\, q_{i,z-2} \ldots q_{i,z}\, q_{i,0}$$

основанном на представлении

$$\mathrm{p_i} = q_{i,z-1}Q^{z-1} + q_{i,z-2}Q^{z-2} + \ldots + q_{i,1}Q^1 + q_{i,0}, \qquad (3.13)$$

где $q_{i,j}$ - базисные числа системы счисления с основанием Q.

Тогда в смешанной системе счисления число N будет записываться в виде

$$N = q_{n,z-1}\, q_{n,z-2} \ldots q_{n,0}\, q_{n-1,z-1} \ldots q_{n-1,0} \ldots q_{0,z-1} \ldots q_{0,0}$$

Подставляя (3.13) в (3.12) и учитывая соотношение $P = Q^z$, получим

$$\begin{aligned} N = {} & q_{n,z-1}Q^{nz+z-1} + q_{n,z-2}Q^{nz+z-2} + \ldots + q_{n,0}Q^{nz} + q_{n-1,z-1}Q^{nz-1} + q_{n-1,0}Q^{nz-1} + \ldots + \\ & + q_{0,z-1}Q^{nz-1} + \ldots + q_{0,z-1}Q^{z-1} + \ldots + q_{0,0}Q^{0} \end{aligned} \qquad (3.14)$$

т.е. разложение числа N по степеням Q. Поэтому запись числа N в Q-ичной системе счисления, соответствующая разложению (3.14), будет иметь вид

$$N = q_{n,z-1}\, q_{n,z-2} \ldots q_{n,0}\, q_{n-1,z-1} \ldots q_{n-1,0} \ldots q_{0,z-1} \ldots q_{0,0}\,.$$

Как видно, эта запись тождественно совпадает с приведенной выше записью числа N в смешанной системе счисления, где каждая очередная группа из z цифр является просто изображением соответствующего коэффициента p_i, в системе счисления с основанием Q.

Все сказанное выше относительно целых чисел автоматически переносится и на случай произвольных чисел. Таким образом, изображение числа x в P-ичной системе счисления в случае $P = Q^z$ является просто сокращенной записью изображения этого же числа x в Q-ичной системе.

Рассмотренное выше свойство некоторых смешанных систем широко используется на практике для сокращенной записи чисел, заданных в системе счисления с небольшим основанием. Для этого в исходной записи числа разряды объединяются вправо и влево от точки в группы некоторой длины (добавляя в случае необходимости левее старшей или правее младшей значащих цифр соответствующее количество нулей), и каждая такая группа записывается одной цифрой другой системы, основание которой равно соответствующей степени исходного основания.

Перевод чисел из одной системы счисления в другую

При решении задач с помощью ЭВМ исходные данные обычно задаются в десятичной системе счисления; в этой же системе, как правило, нужно получить и окончательные результаты. Так как в современных ЭВМ данные кодируются в основном в двоичных кодах, то, в частности, возникает необходимость перевода чисел из десятичной в двоичную систему счисления и наоборот.

При рассмотрении правил перевода чисел из одной системы счисления в другую ограничимся только такими системами счисления, у которых базисными числами являются последовательные целые числа от 0 до P-1 включительно, где P - основание системы счисления.

Задача перевода заключается в следующем. Пусть известна запись числа x в системе счисления с каким-либо основанием P:

$$p_n p_{n-1} \ldots p_1 p_0 p_{-1} p_{-2} \ldots,$$

где p_i - цифры P-ичной системы ($0 \le p_i \le P$-1). Требуется найти запись этого же числа x в системе счисления с другим основанием Q:

$$q_s q_{s-1} \ldots q_1 q_0 q_{-1} q_{-2} \ldots,$$

где q_i - искомые цифры Q-ичной системы ($0 \le q_i \le Q$-1). При этом можно ограничиться случаем положительных чисел, так как перевод любого числа сводится к переводу его модуля и приписыванию числу нужного знака.

При рассмотрении правил перевода нужно учитывать, средствами какой арифметики должен быть осуществлен перевод, т.е. в какой системе счисления должны быть выполнены все необходимые для перевода действия. Условимся считать, что перевод должен осуществляться средствами P-ичной арифметики.

Перевод Q→P. Задача перевода произвольного числа x, заданного в системе счисления с основанием Q, в систему счисления с основанием P сводится к вычислению полинома вида

$$X = q_n Q^n + q_{n-1} Q^{n-1} + \ldots + q_1 Q^1 + q_0 Q^0 + q_{-1} Q^{-1} + \ldots + q_{-m} Q^{-m} \qquad (3.15)$$

Для получения P-ичного изображения выражения (3.15) необходимо все цифры q_i и число Q заменить P-ичными изображениями и выполнить арифметические операции в P-ичной системе счисления.

Заметим, что при переводе следует придерживаться правила сохранения точности изображения числа в разных системах, причем под точностью понимается значение единицы самого младшего (правого) разряда, используемого в записи числа в той или иной системе счисления.

Перевод P→Q. Так как для перевода любого числа достаточно уметь переводить его целую и дробную части, рассмотрим отдельно эти два случая.

1. Перевод целых чисел. Пусть известна запись целого числа N в системе счисления с основанием P и требуется перевести это число в систему счисления с основанием Q. Так как N - целое, то его запись в Q-ичной системе счисления имеет вид

$$N = q_s q_{s-1} \ldots q_1 q_0,$$

где q_i - искомые цифры Q-ичной системы ($0 \le q_i < Q$-1). Для определения q_0 разделим обе части равенства:

$$N = q_s Q^s + q_{s-1} Q^{s-1} + \ldots + q_1 Q^1 + q_0 \qquad (3.16)$$

на число Q, причем в левой части произведем деление, пользуясь правилами P-ичной арифметики (так как запись числа N в P-ичной системе счисления известна), а правую часть перепишем в виде

$$N/Q = q_s Q^s + q_{s-1} Q^{s-1} + \ldots + q_1 Q^1 + q_0 / Q.$$

Приравнивая между собой полученные целые и дробные части (учитывая, что $q_i < Q$):

$$[N/Q] = q_s Q^{s-1} + q_{s-1} Q^{s-2} + \ldots + q_1,$$
$$[N/Q] = q_0/Q.$$

Таким образом, младший коэффициент q_0 в разложении (3.16) определяется соотношением

$$Q_0=Q[N/Q],$$

причем указанные здесь действия на самом деле не выполняются, так как q_0 является просто остатком от деления N на Q.

Положим

$$N_1=[N/Q]= q_sQ^{s-1}+ q_{s-1}Q^{s-2}+\ldots+ q_1 .$$

Тогда N_1 будет целым числом и к нему можно применить ту же самую процедуру для определения следующего коэффициента q_1 и т.д.

Таким образом, при условии что $N_0 = N$, перевод чисел с использованием P-ичной арифметики осуществляется по следующим рекуррентным формулам:

$$q_i = Q[N_i / Q], \qquad (3.17)$$
$$N_{i+1} = [N_i / Q] \ (i=0, 1, 2).$$

Этот процесс продолжается до тех пор, пока не будет получено $N_{i+1}=0$.

Заметим, что поскольку все операции выполняются в системе счисления с основанием P, то в этой же системе будут получены искомые коэффициенты q_i, поэтому их необходимо записать одной Q-ичной цифрой.

2. Перевод дробных чисел. Пусть необходимо перевести в Q-ичную систему счисления правильную дробь x ($0 < x < 1$), заданную в P-ичной системе счисления.

Так как $x < 1$, то число x в Q-ичной системе счисления можно представить в виде полинома

$$x = q_{-1}Q^{-1} + q_{-2}Q^{-2} + \ldots + q_{-m}Q^{-m} +\ldots, \qquad (3.18)$$

где q_{-i} (i = 1, 2, ...) - искомые коэффициенты Q-ичного разложения числа x. Для определения q_{-1} умножим обе части равенства (3.18) на число Q, причем в левой части произведем умножение, пользуясь правилами P-ичной арифметики (так как запись числа x в P-ичной системе счисления известна), а правую часть перепишем в виде

$$xQ = q_{-1} + q_{-2}Q^{-1} + q_{-m}Q^{-m}+\ldots .$$

Приравняем между собой полученные в правой части этого выражения целые и дробные части (учитывая, что $0 < q_i < Q$):

$$[xQ]=q_{-1} ,$$
$$[xQ]=q_{-2}Q^{-1} + \ldots + q_{-m}Q^{-m+1} +\ldots .$$

Таким образом, младший коэффициент q_{-1} в разложении (3.18) определяется соотношением

$$q_{-1} = [x_iQ].$$

Положим,

$$x_1 = [xQ] = q_{-2}Q^{-1} +\ldots+ q_{-m}Q^{-m+1} +\ldots .$$

Тогда x_1 будет правильной дробью и к этому числу можно применить ту же самую процедуру для определения следующего коэффициента q_{-2} и т.д.

Таким образом, при условии, что $x_0=x$, перевод дроби с использованием P-ичной арифметики осуществляется по следующим рекуррентным формулам:

$$q_{-(i+1)} = [x_i Q],$$
$$x_{i+1} = [x_i Q] \ (i=0,1,2,\ldots). \qquad (3.19)$$

Этот процесс продолжается до тех пор, пока не будет получено $x_{i+1}=0$ или не будет достигнута требуемая точность изображения числа.

Замечание. При переводе приближенных дробей из одной системы счисления в другую необходимо придерживаться следующего правила.

Если единица младшего разряда числа x, заданного в P-ичной системе счисления, есть P^{-k}, то в его Q-ичной записи следует сохранить z разрядов после запятой, где z удовлетворяет условию

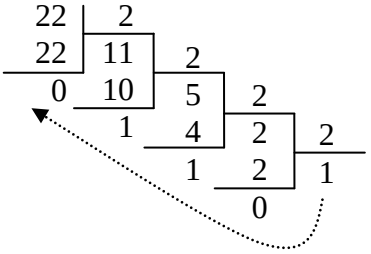
$$Q^{-z} > P^{-k}/2 > Q^{-(z+1)},$$

округляя последнюю оставляемую цифру обычным способом.

Приведем примеры перевода чисел из одной системы счисления в другую методом деления.

Для перевода десятичного числа в двоичную систему его необходимо последовательно делить на 2 до тех пор, пока не останется остаток, меньший или равный 1. Число в двоичной системе записывается как последовательность последнего результата деления и остатков от деления в обратном порядке.

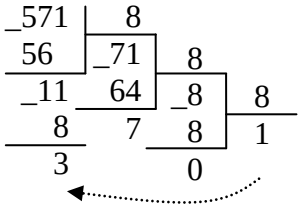
Пример. Число 22_{10} перевести в двоичную систему счисления.



$22_{10} = 10110_2$

Для перевода десятичного числа в восьмеричную систему его необходимо последовательно делить на 8 до тех пор, пока не останется остаток, меньший или равный 7. Число в восьмеричной системе записывается как последовательность цифр последнего результата деления и остатков от деления в обратном порядке.

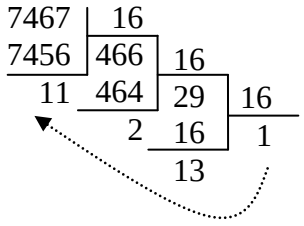
Пример. Число 571_{10} перевести в восьмеричную систему счисления.



$571_{10} = 1073_8$

Для перевода десятичного числа в шестнадцатеричную систему его необходимо последовательно делить на 16 до тех пор, пока не останется остаток, меньший или равный 15. Число в шестнадцатеричной системе записывается как последовательность цифр последнего результата деления и остатков от деления в обратном порядке.

Пример. Число 7467_{10} перевести в шестнадцатеричную систему счисления.



$7467_{10} = 1D2B_{16}$

3.5 Представление данных в компьютере

Представление целых чисел без знака и со знаком

Введем основные понятия на примере 4-битовых машинных слов. Такой размер слова обеспечивает хранение десятичных чисел только от 0 до 15 и поэтому не представляет практического значения. Однако они менее громоздки, а основные закономерности, обнаруженные на примере 4-битовых слов, сохраняют силу для машинного слова любого размера.

Предположим, что процессор ЭВМ способен увеличивать (прибавлять единицу) и дополнять (инвертировать) 4-битовые слова. Например, результатом увеличения слова 1100 является 1101, а результатом дополнения этого слова является 0011. Рассмотрим слово 0000, представляющее десятичное число 0. В результате увеличения содержимое этого слова станет равным 0001, что соответствует десятичному числу 1. Продолжая последовательно увеличивать 4-битовые слова, придем к ситуации, когда, увеличивая слово 1111 (которое представляет десятичное число 15), получим в результате слово 0000, т. е. 111+1 = 0000 (15+1=0), при этом получили неверную арифметическую операцию и вернулись в исходное состояние. Это произошло из-за того, что слово памяти может состоять только из конечного числа битов. Таким образом, числовая система ЭВМ является *конечной* и *цикличной*.

Такой ситуации, приводящей к неверному арифметическому результату, можно избежать, если битовую конфигурацию 1111 принять за код для -1. Тогда 1110 интерпретируется как -2; 1101 – 3 и т.д. до 1000 – 8. Тем самым получили другую числовую систему – со знаком, содержащую как положительные, так и отрицательные числа. В этой системе половина четырехбитовых конфигураций, начинающаяся с единицы, интерпретируется как отрицательные числа, а другая половина, начинающаяся с 0, – как положительные числа или нуль. Поэтому старший бит числа (третий по

счету, если нумерацию битов начинать с нуля справа налево) называется знаковым битом. Числовая система со знаком также конечна и циклична, однако в этом случае арифметически неверный результат даст попытка увеличить число 8 на единицу. Преимущество введения числовой системы со знаком заключается в возможности представления как положительных, так и отрицательных чисел.

Если знаковый бит равен нулю, то значение числа легко вычисляется - игнорируется знаковый бит, а оставшиеся три бита интерпретируются как двоичный код десятичного числа. Например, слово 0110 представляет двоичное число 110, которое равно десятичному числу 6.

Для оценки отрицательного числа нужно изменить его знак. Рассмотрим четырехбитовое число k в системе со знаком. Тогда $-k = (-1-k)+1$, следовательно, для вычисления значения $-k$ необходимо вычесть k из -1 (т.е. из 1111) и затем прибавить 1 (т.е. 0001). Заметим, что операция вычитания всегда возможна, никогда не требует заема и равнозначна операции инвертирования битов вычитаемого. Например, 1111 - 1011 = 0100, здесь в вычитаемом, равном 1011, единицы перешли в нули, а нуль - в единицу. Инвертирование битов в слове называется *дополнением до единицы*. Для определения отрицательного значения числа k надо к его дополнению до единицы прибавить единицу (согласно вышеприведенному равенству). Инвертирование битов в слове с добавлением единицы к младшему биту называется *дополнением до двух*. Например, требуется найти, какое число закодировано в слове 1001. Для этого сначала выполняем операцию инвертирования $1001 \rightarrow 0110$, а затем к полученному результату прибавляем единицу 0110+1 = 0111, что является двоичным кодом числа 7. Таким образом, значением 1001 является отрицательное 7, т.е. -7.

Индикаторы переноса и переполнения

Рассмотрим более подробно ситуацию, приводящую при увеличении четырехбитового числа (т.е. прибавления к нему 1) к неверному арифметическому результату, возникшую из-за конечности числовой системы ЭВМ. В числовой системе без знака эта проблема возникает при увеличении слова 1111, при этом имеет место перенос единицы из знакового бита. В случае системы чисел со знаком перенос из старшего бита дает верный результат: 1111+ 0001 = 0000 (что правильно: -1 + 1 = 0). Но в этой системе увеличение слова 0111 приводит к ошибочной ситуации: 0111 + 1 = 1000 (7 + 1 = -8), при этом имеет место перенос в знаковый бит.

В арифметико-логическом устройстве (АЛУ) процессора ЭВМ содержатся два индикатора - индикатор переноса и индикатор переполнения. Каждый индикатор содержит 1 бит информации и может быть процессором установлен (в этом случае ему придается значение, равное 1) или сброшен (равен 0). Индикатор переноса указывает на перенос из знакового бита, а индикатор переполнения - на перенос в знаковый бит. Таким образом, после завершения операции, в которой происходит перенос в старший бит,

процессор устанавливает индикатор переполнения, если такого переноса нет, то индикатор переполнения сбрасывается. Индикатор переноса обрабатывается аналогичным образом.

Важно то, что после выполнения операции процессором, ЭВМ сигнализирует о состоянии индикаторов и их можно проверить. Если состояние индикаторов указывает на неправильный арифметический результат, то необходимо предпринять меры, исправляющие эту ситуацию, т.е. пользователю ЭВМ предоставляется возможность контролировать правильность выполнения арифметических операций.

Следует иметь в виду, что способ обработки индикаторов процессором (т.е. их установление или сброс) зависит от выполняемой операции, а не просто от того, были или нет переносы. В одних случаях установка индикаторов будет происходить так, как описано выше. В других случаях один или другой индикатор может быть установлен или сброшен независимо от того, происходил ли в действительности перенос в знаковый бит или из него, или есть случаи, когда индикаторы остаются без изменения. Таким образом, правила для условий, при которых индикаторы устанавливаются, сбрасываются или остаются без изменения, зависят от выполняемой операции. При конструировании ЭВМ эти условия учитываются, и аппаратура разрабатывается таким образом, чтобы индикаторы переноса и переполнения давали информацию о правильности выполнения операций в целом ряде обстоятельств.

Например, правильность операции сложения определяется на основании следующих условий:

1. Если машинные слова интерпретируются как числа без знака, то результат сложения двух слов будет арифметически правильным тогда и только тогда, когда не будет переноса из знакового бита.

2. Если машинные слова интерпретируются как числа со знаком, то результат сложения

а) двух положительных чисел будет арифметически правильным тогда и только тогда, когда не будет переноса в знаковый бит;

б) двух отрицательных чисел будет арифметически правильным тогда и только тогда, когда будет происходить перенос в знаковый бит, причем в этой ситуации перенос из знакового бита происходит всегда;

в) отрицательного и положительного чисел всегда будет правильным, а перенос в знаковый бит будет происходить тогда и только тогда, когда будет также происходить перенос из знакового бита.

Таким образом, если в результате выполнения операции сложения происходит перенос из знакового бита, то индикатор переноса устанавливается, если нет, то индикатор переноса сбрасывается, т.е. состояние индикатора переноса, сброшенное или установленное, показывает соответственно правильность или неправильность операции сложения без учета знака. Индикатор переполнения устанавливается, если два складываемых числа имеют один и тот же знак, а результат сложения получается с противоположным знаком; в противном случае он

сбрасывается, т.е. индикатор переполнения устанавливается тогда и только тогда, когда происходит только один из переносов в знаковый бит или из него. Тем самым состояние индикатора переполнения (сброшенное или установленное) может использоваться для определения соответственно правильности или неправильности сложения с учетом знака.

Пример 1. Рассмотрим сложение двух чисел 0101+0011=1000. В результате выполнения операции сложения произошел перенос в знаковый бит, а переноса из знакового бита не было. Таким образом, после завершения этой операции индикатор переноса будет сброшен, а индикатор переполнения установлен. Поэтому если рассматривать эти два числа как целые без знака, то результат является арифметически правильным, так как индикатор переноса сброшен. Если рассматривать числа в системе со знаком, то бит переполнения показывает, что произошло изменение знака (перенос в знаковый бит есть, а из – нет), поэтому арифметический результат неправильный.

Пример 2. В результате сложения 1101+0101=0010 происходит перенос и в знаковый бит, и из знакового бита. Поэтому будет установлен индикатор переноса, а индикатор переполнения сброшен. Следовательно, в системе чисел без знака результат является арифметически неправильным, а в системе чисел со знаком – правильным.

Пример 3. В результате сложения 0011+0010=0101 не происходит переноса ни в знаковый бит, ни из него. Поэтому оба индикатора будут сброшены. Следовательно, в этом случае в обеих числовых системах (без знака и со знаком) результат будет арифметически правильным.

Операцию вычитания можно свести к операции сложения в силу того, что A–B=A+(–B). Таким образом, необходимо только над вычитаемым произвести операцию дополнения до двух и сложить его с уменьшаемым. Например, операция 0011-1001 эквивалентна операции 0011+(0110+1)=0011 + 0111 = 1010. В этом случае в системе чисел без знака (3 - 9 = 10) и со знаком (3-(-7) = -6) результат является арифметически неправильным.

Правильность или неправильность результатов вычитания, так же как и при сложении, зависит от того, происходили (или нет) переносы в знаковый бит или из него. Чтобы понять, как процессор устанавливает индикаторы переноса и переполнения, надо помнить, что вычитание выполняется как сложение: A-B=A+(-B). Если это сложение приводит к переносу из знакового бита, то индикатор переноса сбрасывается, иначе он устанавливается. Следовательно, в случае вычитания индикатор переноса устанавливается обратно тому, как при сложении. Индикатор переполнения устанавливается, если уменьшаемое и вычитаемое имеют противоположные знаки (т.е. имеют разные знаковые биты), а результат вычитания имеет тот же знак, что и вычитаемое, то индикатор переполнения сбрасывается.

Таким образом, состояние индикатора переноса (сброшен или установлен) показывает соответственно на правильность и неправильность вычитания в числовой системе без учета знака, а сброшенный или установленный индикатор переполнения показывает соответственно на

правильность или неправильность вычитания в числовой системе со знаком. При этом индикатор переноса устанавливается тогда и только тогда, когда нет переноса из знакового бита, а индикатор переполнения устанавливается тогда и только тогда, когда был только один перенос в знаковый бит или из него.

Пример 4. В результате выполнения операции вычитания 1001-0011 = =1001+(-0011)=1001+(1100+1)=1001+1101=0110 происходит перенос из знакового бита, а переноса в знаковый бит нет. Следовательно, индикатор переноса будет сброшен, а индикатор переполнения установлен, что указывает на то, что в данном примере в системе без знака результат арифметически правильный, а в системе со знаком – неправильный.

Представление символьной информации в ЭВМ

В отличие от обычной словесной формы, принятой в письменном виде, символьная информация хранится и обрабатывается в памяти ЭВМ в форме цифрового кода. Например, можно обозначить каждую букву числами, соответствующими ее порядковому номеру в алфавите: А - 01, Б - 02, В - 03, Г - 04, ... , Э - 30, Ю - 31, Я - 32. Точно так же можно договориться обозначать точку числом 33, запятую - 34 и т.д. Так как в устройствах автоматической обработки информации используются двоичные коды, то обозначения букв надо перевести в двоичную систему. Тогда буквы будут обозначаться следующим образом: А - 000001, Б - 000010, В - 000011, Г - 000100, ... , Э - 011110, Ю - 011111, Я - 100000. При таком кодировании любое слово можно представить в виде последовательности кодовых групп, составленных из 0 и 1. Например, слово ЭВМ выглядит так: 011110000011001110.

При преобразовании символов (знаков) в цифровой код между множествами символов и кодов должно иметь место взаимно-однозначное соответствие, т.е. разным символам должны быть назначены разные цифровые коды, и наоборот. Это условие является единственным необходимым требованием при построении схемы преобразования символов в числа. Однако существует ряд практических соглашений, принимаемых при построении схемы преобразования исходя из соображений наглядности, эффективности, стандартизации. Например, какое бы число ни назначили коду для знака 0 (не следует путать с числом 0), знаку 1 удобно назначить число, на единицу большее, чем код 0, и т.д. до знака 9. Аналогичная ситуация возникает и при кодировке букв алфавита: код для Б на единицу больше кода для А, а код для В на единицу больше кода для Б и т. д. Таким образом, из соображений наглядности и легкости запоминания целесообразно множества символов, упорядоченных по какому-либо признаку (например, лексико-графическому), кодировать также с помощью упорядоченной последовательности чисел.

Другим важным моментом при организации кодировки символьной информации является эффективное использование оперативной памяти ЭВМ.

Так как общеупотребительными являются примерно 100 знаков (сюда помимо цифр, букв русского и английского алфавитов, знаков препинания, арифметических знаков входят знаки перевода строки, возврата каретки, возврата на шаг и т.п.), то для взаимно-однозначного преобразования всех знаков в коды достаточно примерно сотни чисел. Значение этого выбора заключается в том, что для размещения числа из этого диапазона в оперативной памяти достаточно одного байта, а не машинного слова. Следовательно, при такой организации кодировки достигается существенная экономия объема памяти.

При назначении кодов знакам надо также учитывать соглашения, касающиеся стандартизации кодировки. Можно назначить знаковые коды по своему выбору, но тогда возникнут трудности, связанные с необходимостью обмена информацией с другими организациями, использующими кодировку, отличную от нашей. В настоящее время существует несколько широко распространенных схем кодирования. Например, код BCD (Binary-Coded Decimal) - двоично-десятичный код используется для представления чисел, при котором каждая десятичная цифра записывается своим четырехбитовым двоичным эквивалентом. Этот код может оказаться полезным, когда нужно преобразовать строку числовых знаков, например, строку из числовых знаков «2537» в число 2537, над которым затем будут производиться арифметические действия. Расширением этого кода является EBCDIC (Extended Binary-Coded Decimal Interchange Code) - расширенный двоично-десятичный код обмена информацией, который преобразует как числовые, так и буквенные строки.

В ЭВМ, начиная с типа PDP (или СМ) и по сегодняшний день в IBM PC применяется код ASCII (American Standard Code for Information Interchange) - американский стандартный код обмена информацией. Этот код генерируется некоторыми внешними устройствами (принтером, АЦПУ) и используется для обмена данными между ними и оперативной памятью ЭВМ. Например, когда нажимаем на терминале клавишу G, то в результате этого действия код ASCII для символа G (1000111) передается в ЭВМ. А если надо этот символ распечатать на АЦПУ, то его код ASCII должен быть послан на печатающее устройство.

С развитием техники понадобилось вводить национальные варианты ASCII. Стандарт ISO 646 (ECMA-6) предусматривает возможность размещения национальных символов на месте @ [\ ] ^ ` { | } ~. В дополнение к этому, на месте # может быть размещён £, а на месте $ — ¤. Такая система хорошо подходит для европейских языков, где нужны лишь несколько дополнительных символов. Вариант ASCII без национальных символов называется US-ASCII, или «International Reference Version».

Для некоторых языков с нелатинской письменностью (русского, греческого, арабского, иврита) существовали более радикальные модификации ASCII. Одним из вариантов был отказ от строчных латинских букв — на их месте размещались национальные символы (для русского и греческого — только заглавные буквы). Другой вариант — переключение

между US-ASCII и национальным вариантом «на лету» с помощью символов SO (Shift Out) и SI (Shift In) — в этом случае в национальном варианте можно полностью устранить латинские буквы и занять всё пространство под свои символы.

Впоследствии оказалось удобнее использовать 8-битные кодировки (кодовые страницы), где нижнюю половину кодовой таблицы (0—127) занимают символы US-ASCII, а верхнюю (128—255) — дополнительные символы, включая набор национальных символов. Таким образом, верхняя половина таблицы ASCII до повсеместного внедрения Юникода (UNICODE) активно использовалась для представления локализированных символов, букв местного языка. Отсутствие единого стандарта размещения кириллических символов в таблице ASCII доставляло множество проблем с кодировками (КОИ-8, Windows-1251 и другие). Другие языки с нелатинской письменностью тоже страдали из-за наличия нескольких разных кодировок.

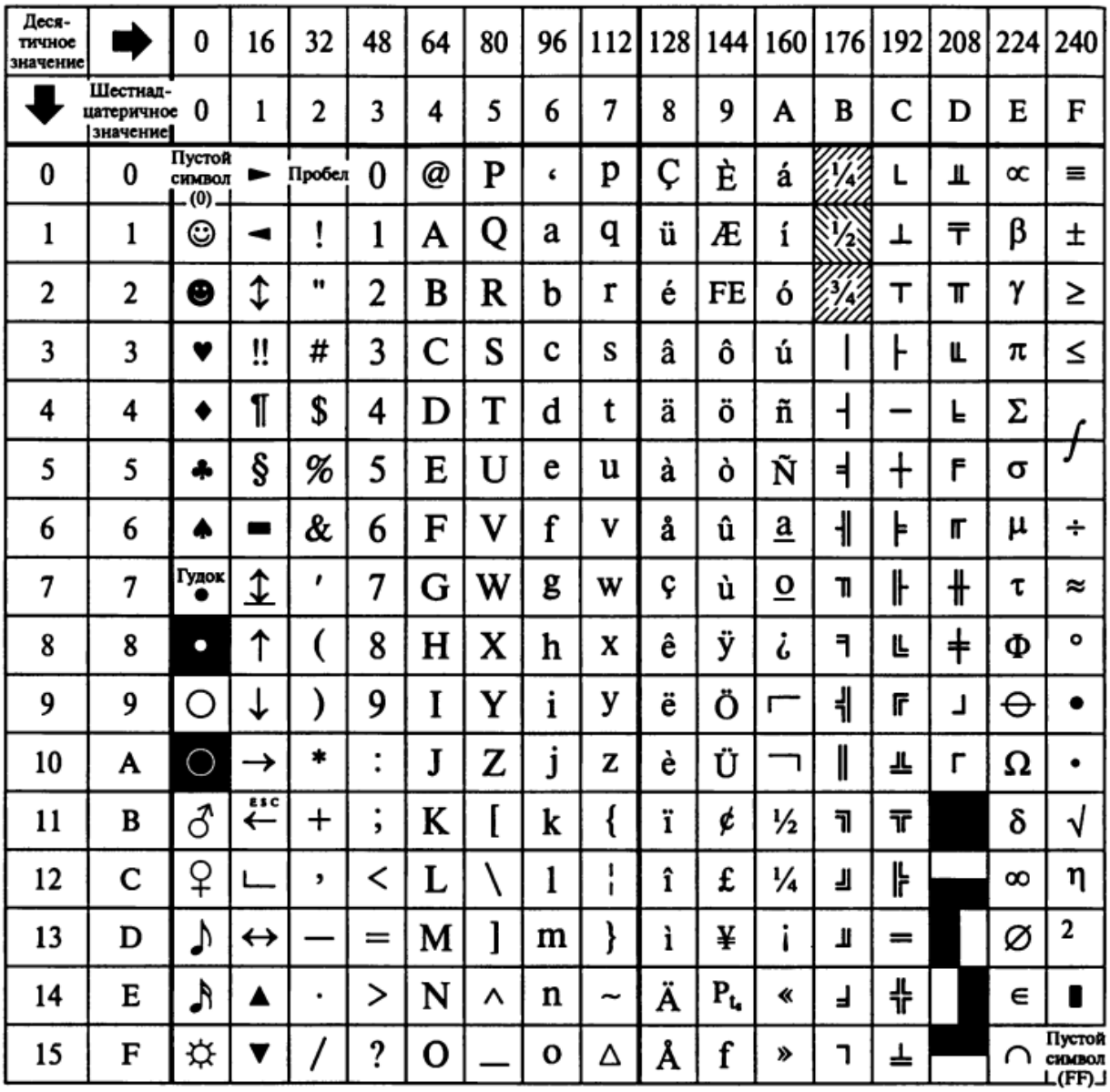


Рисунок 3.21 – Таблица кода ASCII

Отечественной версией кода ASCII являлся код КОИ-7 (двоичный семибитовый код обмена информацией), который совпадает с ним, за исключением букв русского алфавита. Для использования с национальными алфавитами и прочими символами, не входящими в ASCII чаще всего стала применяться старшая половина пространства 8-битных кодов (128–255), позволяющее использовать до 128 дополнительных символов, чего достаточно для большинства европейских языков.

КОИ-8, KOI8 — восьмибитовая ASCII-совместимая кодовая страница, разработанная для кодирования букв кириллических алфавитов. Разработчики КОИ-8 поместили символы русского алфавита в верхней части кодовой таблицы таким образом, что позиции кириллических символов соответствуют их фонетическим аналогам в английском алфавите в нижней части таблицы. Это означает, что если в тексте, написанном в КОИ-8, убирать восьмой бит каждого символа, то получается «читаемый» текст, хотя он и написан латинскими символами. Например, слова «Русский Текст» превратились бы в «rUSSKIJ tEKST». Как побочное следствие, символы кириллицы оказались расположены не в алфавитном порядке.

Существует несколько вариантов кодировки КОИ-8 для различных кириллических алфавитов, расширяющие определённые коды (общий диапазон 192—255 с 32 русскими буквами в двух регистрах остаётся неизменным во всех вариантах). Русский алфавит описывается в кодировке KOI8-R, украинский — в KOI8-U.

KOI8-R стал фактически стандартом для русской кириллицы в 1990-х годах в юникс-подобных операционных системах и электронной почте.

Юникод (*Unicode*) — стандарт кодирования символов, позволяющий представить знаки практически всех письменных языков. Стандарт состоит из двух основных разделов: универсальный набор символов (*UCS, universal character set*) и семейство кодировок (*UTF, Unicode transformation format*). Универсальный набор символов задаёт однозначное соответствие символов кодам - элементам кодового пространства, представляющим неотрицательные целые числа. Семейство кодировок определяет машинное представление последовательности кодов UCS.

Коды в стандарте Юникод разделены на несколько областей. Область с кодами от U+0000 до U+007F содержит символы набора ASCII с соответствующими кодами. Далее расположены области знаков различных письменностей, знаки пунктуации и технические символы. Часть кодов зарезервирована для использования в будущем.[7] Под символы кириллицы выделены области знаков с кодами от U+0400 до U+052F, от U+2DE0 до U+2DFF, от U+A640 до U+A69F

Универсальная система кодирования (Юникод) представляет собой набор графических символов и способ их кодирования для компьютерной обработки текстовых данных.

Графические символы - это символы, имеющие видимое изображение. Графическим символам противопоставляются управляющие символы и

символы форматирования.

Графические символы включают в себя следующие группы:

- буквы, содержащиеся хотя бы в одном из обслуживаемых алфавитов;
- цифры;
- знаки пунктуации;
- специальные знаки (математические, технические, идеограммы и пр.);
- разделители.

Юникод — это система для линейного представления текста. Символы, имеющие дополнительные над- или подстрочные элементы, могут быть представлены в виде построенной по определённым правилам последовательности кодов (составной вариант, composite character) или в виде единого символа (монолитный вариант, precomposed character).

Форматы данных

Под *данными* будем понимать информацию, представленную в виде, пригодном для обработки автоматическими средствами, например, в двоичном коде. Формат представления данных в памяти ЭВМ зависит от ее архитектуры.

Данные, обрабатываемые ЭВМ, делятся на три группы: логические коды, числа с фиксированной запятой и числа с плавающей запятой.

Представление логических кодов. Логические коды могут размещаться в отдельных байтах и в словах. Для их представления используются все разряды: для байта от 0-го до 7-го, для слова - от 0-го до 15-го. Логическими кодами могут быть представлены символьные величины, числа без знака и битовые величины.

Символьные величины задаются в коде ASCII (КОИ-7), каждый символ занимает один байт, разряд 7 которого всегда содержит 0. Символы строки размещаются в последовательно-адресуемых байтах оперативной памяти. Например, символьная строка ABCDE (коды ASCII: A- 101_8,B- 102_8,C- 103_8, D- 104_8,E- 105_8), первый знак которой помещается в ячейку с адресом 1000 (адреса представлены в 8-ричной системе счисления), размещается в оперативной памяти следующим образом:

1001	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	1	**1000**
1003	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1	**1002**
1005									0	1	0	0	0	1	0	1	**1004**

Рисунок 3.22 – Размещение информации в памяти

Числа без знака имеют диапазон представления от 000 до 377_8 -для байта, от 000000 до 177777_8 - для слова. Битовые величины задают значения отдельных разрядов байта или слова.

Представление чисел в формате с фиксированной запятой. Числа с фиксированной запятой могут занимать байт или слово. Если число с

фиксированной запятой занимает байт, то для его представления используются разряды с 0-го по 6-й. Разряд 7 называется знаковым. При размещении числа с фиксированной запятой в слове для его представления используются разряды с 0-го по 14-й. Знак числа содержится в разряде 15. Значения знакового разряда: 0 - для положительных чисел; 1 - для отрицательных чисел.

Отрицательные числа в формате с фиксированной запятой представляются в дополнительном коде (посредством операции дополнения до 2-х).

Таблица 3.14 – Примеры представления чисел с фиксированной запятой

	Байт		Слово	
Число	Восьмеричный код	Двоичный код	Восьмеричный код	Двоичный код
+5	005	00000101	000005	0000000000000101
+63	077	00111111	000077	0000000000111111
-5	373	11111011	177773	1111111111111011
-63	315	11001101	177715	1111111111001101
0	000	00000000	000000	0000000000000000

Диапазон представления чисел с фиксированной запятой: для байта - от - 128_{10} до $+127_{10}$; для слова - от -32768_{10} до $+32767_{10}$. При выполнении операций над числами, представленными в формате с фиксированной запятой, они масштабируются таким образом, чтобы каждое число лежало в интервале (-1, +1). Другими словами, в этом случае ЭВМ оперирует только с числами, по модулю не превосходящими единицы. При этом необходимо следить за тем, чтобы в процессе операций результат не получился большим, чем 2^k - 1, где k - число разрядов, отведенных для представления в машине. Такая опасность есть при выполнении операций сложения и деления. Опасность представляют также операции вычитания и умножения. При вычитании может получиться так, что разность станет числом меньшим, чем представляется в машине, и результат исчезнет. При многократном умножении (из-за того, что умножаются числа, меньшие единицы) может произойти то же самое. Поэтому при использовании формата представления чисел с фиксированной запятой приходится следить как за случаями возможного переполнения разрядной сетки машины, так и за случаями, связанными с появлением машинного нуля. Необходимость постоянно следить за тем, чтобы числа в машине не вышли за пределы интервала (-1, +1), а также неизбежное в таких устройствах накопление абсолютной погрешности вычислений из-за перемасштабирования, при котором цифры младших разрядов (а именно в них накапливается абсолютная погрешность) передвигаются в старшие разряды, привели к тому, что в универсальных ЭВМ представление чисел с фиксированной запятой практически перестало применяться. Оно сохраняется в специализированных вычислительных системах, где диапазон изменения чисел заранее проанализирован, в

некоторых микропроцессорах и микро-ЭВМ.

Представление чисел в формате с плавающей запятой. Любое вещественное число x, представленное в системе счисления с основанием N, можно записать в виде

$$x=\pm mN^{\pm p},$$

где m - мантисса, p - характеристика (или порядок) числа.

Если $|m| < 1$, то запись числа называется *нормализованной слева*. Следующие примеры показывают, как можно представить любое число в форме с плавающей запятой:

а) в десятичной системе счисления

$372{,}95 = 0{,}37295\times10^{3}$;

$25 = 0{,}025\times10^{3} = 0{,}25\times10^{2}$;

$0{,}0000015 = 0{,}15\times10^{-5} = 0{,}015\times10^{-4}$;

б) в двоичной системе счисления

$11010{,}1101 = 0{,}0110101101\times2^{6}$;

$0{,}011011 = 0{,}11011\times2^{-1}$;

$0{,}1 = 0{,}1\times2^{0}$

(здесь порядок определяет, на сколько разрядов необходимо осуществить сдвиг относительно запятой).

Из этих примеров видно, что представление чисел в форме с плавающей запятой неоднозначно. В ЭВМ с целью минимизации погрешности при вычислениях и эффективного использования памяти применяют процедуру нормализации справа.

Число называют *нормализованным справа*, если после запятой в мантиссе стоит не нуль. В дальнейшем под нормализацией числа будем понимать нормализацию справа.

Нормализованное число одинарной точности, представленное в формате с плавающей запятой, записывается в память следующим образом: знак числа - в бите 15 первого слова (0 - для положительных и 1 - для отрицательных чисел); порядок размещается в битах 7 -14 первого слова, а мантисса занимает остальные 23 бита в двух словах. Нормализованное число двойной точности записывается в четыре слова памяти и отличается от представления чисел одинарной точностью только тем, что продолжение мантиссы размещается в следующих за первым трех последовательных словах памяти, а всего под мантиссу в этом случае отводится 55 бит.

Порядок числа с плавающей запятой изменяется в диапазоне от -128_{10} (200_8) до $+127_{10}$ (177_8) и запоминается увеличенным на 128_{10} (200_8). Такой способ представления порядка называется *смещенным*.

Следует иметь в виду, что, хотя для мантиссы отведены 23 разряда -для чисел одинарной точности и 55 разрядов - для чисел двойной точности, в операциях участвуют 24 и 56 разрядов соответственно, так как старший разряд мантиссы нормализованного числа не хранится, т.е. имеет место так

называемый скрытый разряд. Однако при аппаратном выполнении операций этот разряд автоматически восстанавливается и учитывается при выполнении операций. Порядок числа также учитывает скрытый старший разряд мантиссы.

Также заметим, что нормализованная мантисса в двоичной системе счисления всегда представляется десятичным числом т, лежащим в диапазоне $0{,}5 \le m < 1$.

Примеры представления чисел с плавающей запятой:

1)$0{,}1_{10} = 0{,}0(6314)_8 = 0{,}000(1100)_2 = 0{,}(1100)_2 \times 2^{-3}$; $-3_{10} = (-3 + 200)_8 = 175_8 = 01111101_2$.

Заметим, что в этом примере мантисса представлена бесконечной периодической дробью, поэтому последний учитываемый разряд мантиссы округляется.

2) $-49{,}5_{10} = -61{,}4_8 = -110001{,}100_2 = -0{,}1100011_2 \times 2^6$; $6_{10} = (6 + 200)_8 = 206_8 = 10000110_2$.

При выполнении арифметических операций над числами, представленными в формате с плавающей запятой, надо отдельно выполнять их для порядков и мантисс. При алгебраическом сложении чисел надо сначала уравнять порядки слагаемых. При умножении порядки надо складывать, а мантиссы перемножать. При делении из порядка делимого вычитают порядок делителя, а над мантиссами совершают обычную операцию деления. После выполнения операций необходимо провести нормализацию результата, если это необходимо, что приводит к изменению порядков, так как каждый сдвиг на один разряд влево соответствует уменьшению порядка на единицу, а сдвиг вправо - увеличению его на единицу. Введение термина «*плавающая запятая*» как раз и объясняется тем, что двоичный порядок, определяющий фактическое положение запятой в изображении числа, корректируется после выполнения каждой арифметической операции, т.е. запятая в изображении числа плавает (изменятся ее положение) по мере изменения данной величины. А в изображении чисел с фиксированной запятой - запятая жестко зафиксирована в определенном месте.

Арифметические операции с числами в форме плавающей запятой намного сложнее таких же операций для чисел с фиксированной запятой. Но зато плавающая запятая позволяет производить операции масштабирования автоматически в самой машине и избавляет от накопления абсолютной погрешности при вычислениях (хотя не избавляет от накопления относительной погрешности).

Контрольные вопросы

1. Что такое дискретный автомат?
2. Расскажите принцип работы дискретного автомата.
3. Расскажите принцип работы логического автомата.
4. Как работает автомат с конечной памятью?
5. Опишите работу машины Тьюринга
6. Как обеспечивается контроль работы автомата?
7. Какие методы кодирования информации вы знаете?
8. Что такое коды Хэмминга?
9. Какие системы счисления вы знаете? В чем суть каждой из них?
10. Как представляется символьная информация в ЭВМ?
11. Перечислите известные вам форматы данных. В чем их различие?

4. ПРИКЛАДНАЯ ИНФОРМАТИКА

4.1 Автоматизация деятельности на основе алгоритмизации

Автоматизация сопровождает человеческое общество с момента его зарождения. Она внутренне присуща его развитию. В методологии ее определяют как замещение процессов человеческой деятельности процессами технических устройств. С каждым новым открытием, человек снимал с себя какую-нибудь обязанность и перекладывал ее на подручные средства, на животных, потом на машины.

Сегодня любое предприятие имеет дело с потоками различной информации, которые нуждаются в быстрой и оперативной обработке. Количество информации зависит в основном от размера предприятия и вида деятельности, чем больше предприятие, тем больше объем и уровень сложности обрабатываемой информации. Огромную помощь здесь оказывают современные компьютерные информационные технологии, профессионально разработанная компьютерная информационная система может существенно облегчить жизнь бухгалтерии и руководителям, позволит вести оперативный учет на предприятии быстро и точно, предоставит широкие возможности анализа, автоматизировав учетные операции, избавит от огромного количества лишней бумаги.

Проектирование информационной системы является, пожалуй, самым важным элементом автоматизации деятельности предприятия. Правильно спроектировать систему означает обеспечить большую часть успеха всего проекта автоматизации. Очень частой ошибкой является внедрение информационной системы при отсутствии какой-либо четко сформулированной системы управления. То есть выражение «создать систему управления» воспринимается как «внедрить нечто компьютерное». Нужно четко осознавать, что система управления первична, а уже создание информационной системы на ее основе, или, попросту говоря, ее реализация в компьютерном виде - вторична.

Многие компании верят в то, что одна только автоматизация приведет к улучшению финансово-экономической ситуации, и начинают усилия по реализации информационных систем непосредственно с автоматизации, пропуская критические шаги понимания и упрощения своих бизнес-процессов. Но нередко эти процессы настолько неупорядочены, что, в общем, создают впечатление хаоса на предприятии. Очевидно, что автоматизировать хаос попросту невозможно. Поэтому прежде чем создавать информационную систему следует пересмотреть систему управления в организации. Изменение бизнес-процессов называют *реинжинирингом* (business processes re-engineering). Так, для начала нужно упорядочить схему бизнес-процессов и систему управления организации в целом:

- определиться с организационной штатной структурой;
- разработать механизм финансово-экономического управления компанией (в том числе определить центры ответственности);

- произвести выделение основных технологических потоков (процессов);
- разработать механизмы организационного управления технологическими потоками;
- на основании созданных механизмов управления сформировать технологию финансового анализа и управления деятельностью технологических потоков.

Если применять вышеперечисленные технологии, будет значительно легче разработать информационную систему. Однако часто приходится упрощать бизнес-процессы на предприятии для того, чтобы было проще описать их на языке компьютеров.

Организация - это набор правил и процедур. Информационная система – это тоже набор правил и процедур, поэтому следует понимать какие инструкции и процедуры какими заменить. Не следует также забывать о человеческом факторе при создании информационной системы. Во-первых, именно людям придется работать с системой - одна работать она в любом случае не сможет. Во-вторых, служащие могут улучшить (или упростить) процессы, с которыми они ежедневно встречаются. Автоматизация должна происходить только после того, как служащие поймут сущность процесса и примут решение о необходимости автоматизации.

После проведения формирования четкой системы управления, начинается непосредственно процесс проектирования информационной системы. Важно, чтобы в проектировании системы участвовали по возможности все сотрудники, которые будут с ней работать. Это позволит определить небольшие особенности и частные потребности в работе каждого отдела организации, поскольку только пользователи будущей системы лучше всего знают, что им нужно.

В проектировании информационной системы также должны участвовать ее разработчики, то есть те, кто будет ее создавать. К выбору разработчика информационной системы нужно подходить очень осторожно. Основными критериями в выборе разработчика являются опыт работы в области создания информационных систем, количество успешно внедренных данной компанией систем на российских предприятиях.

Финансовый менеджер и руководство предприятия должны относиться к автоматизации как к проекту, то есть определить все стадии, характеристики, временные рамки и бюджет. Основными этапами работы над проектом по автоматизации являются:

1. Проведение обследования с целью описания бизнес-процессов организации.
2. Разработка технического задания на систему автоматизации.
3. Разработка технического проекта системы.
4. Разработка системы (иногда называемая настройкой).
5. Различные стадии и этапы внедрения, опытной и промышленной эксплуатации.
6. Выполнение доработок в соответствии с изменившимися

потребностями организации.

Результатом проектирования системы является строго формализованное описание, как объекта ее автоматизации, так и ее самой – это и есть алгоритм деятельности предприятия, а значит и деятельности людей, которые на нем трудятся.

4.2 Методы автоматизации бизнес-процессов

Долгое время автоматизация учреждений в России осуществлялась в виде различного рода подсистем АСУ, основанных на базах данных (кадры, канцелярия, бухгалтерия, зарплата, контроль исполнения и др.) Не умаляя значимости этих подсистем, заметим, что они охватывали лишь до 15-20 % общего объема информации, циркулирующей в учреждении.

Нужды по электронной обработке документов удовлетворялись применением *функциональных пакетов* (редакторов текста и электронных таблиц) и интегрированных пакетов программ Microsoft Office, Perfect Office, Lotus Smart Suite. Эти средства оказались недостаточными для управления огромными потоками бумажных и электронных документов, циркулирующих как внутри одного предприятия, так и между ними. В целом такой подход грешил отсутствием комплексности в автоматизации делопроизводства и управления документооборотом.

В настоящее время развитие информационных технологий привело к появлению методов и средств, обеспечивающих интегрированные решения по автоматизации организации, позволяющие автоматизировать ручные операции и поиск документов, автоматически передавать и отслеживать перемещение документов и контролировать выполнение поручений, связанных с документами.

Рассмотрим основные методы автоматизации бизнес-процессов. Современные организации представляют собой совокупность подразделений, филиалов и отделов, обменивающихся между собой информацией и выполняющих отдельные части общей работы. *Основными фазами жизни неструктурированной информации в организации* являются:

- ввод информации в систему,
- хранение, навигация, поиск и фильтрация документов,
- коллективная работа с документами,
- вывод информации из системы.

Существуют различные способы *ввода информации в систему*. Это, прежде всего, *сканирование документов* и сохранение их в виде графических образов. В системах первого поколения графические образы введенных документов идентифицируются с помощью ключевых слов для последующего поиска необходимой информации (например: система SoftSolutions). Позднее стала применяться *технология оптического*

распознавания символов OCR (Optical Character Recognition). После сканирования и ввода документа в систему его графический образ "переводится" в текст, затем следует исправление ошибок распознавания.

При массовом ручном вводе однотипных документов используются *электронные формы*, которые обеспечивают структуризацию документа путем выделения частей текста и добавления полей (атрибутов), что позволяет упростить заполнение документов и выполнить необходимые вычисления. Информация в офис может поступать и путем *импорта файлов* с магнитных носителей или *по телекоммуникациям* (факсы, сообщения электронной почты и т.п.).

Ввод информации сопровождается *классификацией* документов путем задания атрибутов и ключевых слов, аннотированием их содержания. Для ускорения последующего контекстного поиска производится *полнотекстовое индексирование документов*.

Важное значение для организации эффективного управления неструктурированными документами имеют методы *хранения информации, навигации, поиска и фильтрации документов.*

Документы могут храниться просто в *файловой системе,* и при этом *система каталогов* служит средством группирования и навигации в хранилище документов. В современных ОС есть возможность задания длинных имен каталогов и файлов в качестве названий папок и документов, а также имеются соответствующие средства поиска файлов по их параметрам.

Ряд систем, основанных на электронной почте, хранят документы *в почтовых ящиках* в виде *почтовых сообщений с присоединенными файлами*. Навигация в хранилище упрощается с помощью *вложенных папок личного и коллективного пользования.* Однако в таких системах поиск и фильтрация ограничены лишь отбором и сортировкой документов по атрибутам и тексту почтового сообщения.

Специфический метод хранения реализован в пакете Lotus Notes в виде так называемой *базы документов.* База документов может хранить как однотипную так и разнотипную информацию в виде одного файла. Документы допускают внутреннюю структуризацию на основе *формуляров* путем выделения и добавления полей в документе. Навигацию в базе документов упрощает наличие *страниц баз документов и категорий документов.* Почтовые сообщения также хранятся в виде базы документов, файлы произвольного вида допускается присоединять к текстовым документам.

Многие современные системы электронных документов используют в дополнение к файловой системе так называемые *библиотеки документов*, содержащие в БД карточки документов с атрибутами и ключевыми словами. Для логической группировки документов применяются *папки*. Поиск и фильтрация документов производится по запросам на основе *контекстного поиска*: по атрибутам, по ключевым словам и по полному содержанию текста на основе индекса. При использовании *механизма*

четкого поиска (например, DOCS OPEN) в запросе не должно быть орфографических ошибок, а в тексте документа - ошибок распознавания.

На основе нейронных сетей и искусственного интеллекта реализована технология *нечеткого* поиска по полному содержанию документа (например, технология адаптивного распознавания образов APRP). Нечеткий поиск не требует полного соответствия искомых фраз с содержимым документов, кроме того, исключает потребность в исправлении ошибок после распознавания текста. Система поиска всегда выдает пользователю ответ, наилучшим образом согласованный с терминами или фразами запроса.

Фирмы-производители реляционных СУБД (в частности ORACLE) проповедуют другие схемы хранения - *текстовые и универсальные БД*. Тексты документов хранятся в *символьных полях переменной длины,* расширенные средства SQL-поиска позволяют формировать смешанные запросы для поиска по атрибутам и контекстного поиска, а дополнительные функции обеспечивают обработку текста. Для хранения произвольной информации, в том числе мультимедиа, можно использовать *поля бинарных объектов большой длины BLOB* и/или *гипертекст*. СУБД, расширенные для поиска и обработки такой информации, образуют универсальные сервера БД. Другой способ хранения документов произвольного содержания реализуют *объектно-ориентированные БД* (например, Informix Illustra).

Феномен *распределенного гипертекста* составляет основу широко внедряемой Web- технологии. Хранилище информации представляет собой совокупность *гипертекстовых страниц*, распределенных по узлам сети Internet или корпоративной сети (Intranet). Каждая страница размещается в отдельном файле и представляет собой текст, размеченный с помощью языка HTML. Структуризация документа осуществляется путем форматирования, выделения полей, создания форм для диалогового заполнения документа и организацией внутренних гипертекстовых ссылок. Допускается создание *гипермедиа* включением любой мультимедиа-информации (растровая графика, аудио, видео). Навигация по хранилищу гипертекста осуществляется с помощью внешних *гипертекстовых ссылок URL* на документы, расположенные на различных узлах сети (Web-серверах). Кроме того, для определения местонахождения документов служит *контекстный поиск*. Для ускорения поиска информации в "паутине" применяются специальные программы-роботы, сканирующие Web- сервера и строящие некое подобие индекса. Использование гипертекста позволяет создать информационную инфраструктуру территориально распределенного офиса и упростить диалоговый интерфейс пользователя, что наиболее важно при разработке информационных приложений для руководителей. Организация и автоматизация в офисе *коллективной работы с документами* строятся на технологиях *groupware* и *workflow.*

Технологии groupware ориентированы на небольшие рабочие группы, характеризуются поддержкой выполнения одной коллективной задачи и отсутствием структуризации в организации работ. Поддержка

ограничивается обеспечением коллективного доступа к информации с помощью различных *методов доступа*:

- сетевой доступ к файлам и базе данных;
- локальная и глобальная электронная почта (включая конференции и дискуссии);
- терминальный доступ, пересылка файлов и электронная доска объявлений;
- просмотр и интерпретация гипертекста (гипермедиа).

Нужно отметить, что Web-технологии помимо гипертекстового протокола HTTP включают в себя ряд других методов доступа.

При коллективной работе важно наличие *блокировок* для разрешения конфликтов при совместном использовании ресурсов*, санкционирование доступа* по идентификаторам и паролям, защита информации с помощью *прав доступа*. Дополнительный уровень безопасности обеспечивается методами и средствами *шифрации и электронной подписи*.

Технологии класса workflow служат для автоматизации документооборота в средних и крупных офисах и для них характерно:

- поддержка многопользовательской работы с несколькими задачами одновременно;
- четкая структуризация выполнения работ по ролям и документам с контролем исполнения.

Деловой процесс формализуется как совокупность состояний и переходов, необходимых для описания взаимодействия, как минимум двух субъектов (в частном случае сотрудников предприятия) для достижения выполнения заранее заданного условия. Частным случаем такого взаимодействия является простая пересылка документа из точки в точку.

Одной из реализаций технологии workflow является так называемая "система графов", где каждый шаг представляет собой вектор и отражает движение задания, связанного с документом, или просто передвижения документа от одного субъекта к другому. При этом на человека, отвечающего за правильность функционирования схемы, ложится ответственность учета всевозможных непредвиденных (или отказных) ситуаций, которые могут возникнуть на пути движения документа. Другая реализация основывается на понятии "цикл" ("loop"). В этом случае подразумевается, что наименьшим элементом в схеме взаимодействия является цикл, учитывающий всю гамму взаимодействия между двумя произвольными субъектами. При этом система сама отслеживает замкнутость процесса и, в случае ошибки, указывает место некорректности с указанием ее причины, после чего прекращается генерация нового процесса.

Регламентации взаимоотношений субъектов документооборота дополняется заданием безусловной и условной маршрутизация документов (по электронной почте) и времен обработки документа для контроля и учета исполнения.

Обработка информации базируется на методах и средствах автоматизации организации:

- обработка текста,
- электронные таблицы,
- деловая и презентационная графика,
- планирование работ и совещаний,
- генерация отчетов из базы данных,
- мультимедиа.

Для комплексирования разных видов информации и интеграции пакетов программ используются несколько методов, среди которых центральное место занимает методы *OLE для связывания и встраивания объектов*. OLE (англ. *Object Linking and Embedding*) — технология связывания и внедрения объектов в другие документы и объекты, разработанная корпорацией Майкрософт. В 1996 году Microsoft переименовала технологию в ActiveX.

OLE позволяет передавать часть работы от одной программы редактирования к другой и возвращать результаты назад. Например, установленная на персональном компьютере издательская система может послать некий текст на обработку в текстовый редактор, либо некоторое изображение в редактор изображений с помощью OLE-технологии.

OLE используется при обработке составных документов, может быть использована при передаче данных между различными несвязанными между собой системами посредством интерфейса переноса, а также при выполнении операций с буфером обмена. Идея внедрения широко используется при работе с мультимедийным содержанием на веб-страницах (пример — Веб-ТВ), где используется передача изображения, звука, видео, анимации в страницах HTML либо в других файлах, также использующих текстовую разметку.

Вывод информации осуществляют путем печать документов, публикация их на Web- серверах, в общих почтовых папках и электронных досках объявлений или рассылки по телекоммуникациям.

4.3 Базовые понятия и технологии управления данными

Понятие базы данных

Развитие вычислительной техники и появление емких внешних запоминающих устройств прямого доступа предопределило интенсивное развитие автоматических и автоматизированных систем разного назначения и масштаба, в первую очередь заметное в области бизнес-приложений. Такие системы работают с большими объемами информации, которая обычно имеет достаточно сложную структуру, требует оперативности в обработке, часто обновляется и в то же время требует длительного хранения. Примерами таких систем являются автоматизированные системы управления предприятием, банковские системы, системы резервирования и продажи

билетов и т. д. Другими направлениями, стимулировавшими развитие, стали, с одной стороны, системы управления физическими экспериментами, обеспечивающими сверхоперативную обработку в реальном масштабе времени огромных потоков данных от датчиков, а с другой — автоматизированные библиотечные информационно-поисковые системы.

Это привело к появлению новой информационной технологии интегрированного хранения и обработки данных — *концепции баз данных,* и основе которой лежит механизм предоставления обрабатывающей программе из всех хранимых данных только тех, которые ей необходимы, и в форме, требуемой именно этой программе. При этом сама форма (структура данных и форматы полей, входящих в эту структуру) описывается на логическом, т.е. «видимом» из программы, уровне. Более того, поскольку различные программы могут по-разному «видеть» (а следовательно, и использовать) одни и те же данные, то система должна сделать «прозрачными» для программы все данные, кроме тех, которые для нее являются «своими».

Система баз данных (БД) — это система специально организованных данных, программных, языковых, организационных и технических средств, предназначенных для централизованного накопления и коллективного многоцелевого использования данных.

Под *базой данных (БД)* обычно понимается именованная совокупность данных, отображающая состояние объектов и их отношений в рассматриваемой предметной области. Характерной чертой баз данных является *постоянство:* данные *постоянно* накапливаются и используются; состав и структура данных, необходимых для решения тех или иных прикладных задач, обычно *постоянны* и стабильны во времени; отдельные или даже все элементы данных, могут меняться — но и это есть проявление постоянства — *постоянная* актуальность.

Система управления базами данных (СУБД) — это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Иногда в составе системы баз данных выделяют *архивы.* Основанием для этого является особый режим использования данных, когда только часть данных находится под оперативным управлением СУБД. Все остальные данные (собственно архивы) обычно располагаются на носителях, оперативно не управляемых СУБД. Одни и те же данные в разные моменты времени могут входить как в базы данных, так и в архивы. Система баз данных может не иметь архивов, но если они есть, то в состав такой системы может входить и система управления архивами.

Проблемы совместного использования данных и периферийных устройств компьютеров и рабочих станций быстро породили модель вычислений, основанную на концепции файлового сервера — она создает основу для коллективной обработки, сохраняя простоту работы с персональным компьютером, позволяет совместно использовать данные и периферию.

В этом смысле главной отличительной чертой баз данных является использование централизованной системы управления данными, причем как на уровне файлов, так и на уровне элементов данных. Централизованное хранение совместно используемых данных приводит не только к сокращению затрат на создание и поддержание данных в актуальном состоянии, но и к сокращению избыточности информации, упрощению процедур поддержания непротиворечивости и целостности данных.

Эффективное управление внешней памятью является основной функцией СУБД. Эти обычно специализированные средства настолько важны с точки зрения эффективности, что при их отсутствии система просто не сможет выполнять некоторые задачи уже потому, что их выполнение будет занимать слишком много времени. При этом ни одна из таких специализированных функций, как построение индексов, буферизация данных, организация доступа и оптимизация запросов, не является видимой для пользователя и обеспечивает независимость между логическим и физическим уровнями системы: прикладной программист не должен писать программы индексирования, распределять память на диске и т. д.

Развитие теории и практики создания информационных систем, основанных на концепции баз данных, создание унифицированных методов и средств организации и поиска данных позволяют хранить и обрабатывать информацию о все более сложных объектах и их взаимосвязях, обеспечивая многоаспектные информационные потребности различных пользователей. Основные требования, предъявляемые к системам баз данных, можно сформулировать следующим образом.

Многократное использование данных: пользователи должны иметь возможность использовать данные различным образом.

Простота: пользователи должны иметь возможность легко узнать и понять, какие данные имеются в их распоряжении.

Легкость использования: пользователи должны иметь возможность осуществлять (процедурно) простой доступ к данным, при этом все сложности доступа к данным должны быть скрыты в самой системе управления базами данных.

Гибкость использования: обращение к данным или их поиск должны осуществляться с помощью различных методов доступа.

Быстрая обработка запросов на данные: запросы на данные, в том числе незапланированные, должны обрабатываться с помощью высокоуровневого языка запросов, а не только прикладными программами, написанными с целью обработки конкретных запросов (разработка таких программ в каждом конкретном случае связана с большими затратами времени). Пользователь должен иметь возможность кратко выразить нетривиальные запросы (в нескольких словах или несколькими нажатиями клавиш мыши). Это означает, что средство формулирования должно быть достаточно «декларативным», т. е. упор должен быть сделан на «что», а не на «как». Кроме того, средство обработки запросов не должно зависеть от приложения, т. е. оно должно работать с любой возможной базой данных.

Язык взаимодействия конечных пользователей с системой должен обеспечивать конечным пользователям возможность получения данных без использования прикладных программ.

База данных — это основа для будущего наращивания прикладных программ: базы данных должны обеспечивать возможность быстрой и дешевой разработки новых приложений.

Сохранение затрат умственного труда: существующие программы и логические структуры данных (на создание которых обычно затрачивается много человеко-лет) не должны переделываться при внесении изменений в базу данных.

Наличие интерфейса прикладного программирования: прикладные программы должны иметь возможность просто и эффективно выполнять запросы на данные; программы должны быть изолированы от расположения файлов и способов адресации данных.

Распределенная обработка данных: система должна функционировать в условиях вычислительных сетей и обеспечивать эффективный доступ пользователей к любым данным распределенной БД, размещенным в любой точке сети.

Адаптивность и расширяемость: база данных должна быть настраиваемой, причем настройка не должна вызывать перезаписи прикладных программ. Кроме того, поставляемый с СУБД набор предопределенных типов данных должен быть расширяемым — в системе должны иметься средства для определения новых типов и не должно быть различий в использовании системных и определенных пользователем типов.

Контроль за целостностью данных: система должна осуществлять контроль ошибок в данных и выполнять проверку взаимного логического соответствия данных.

Восстановление данных после сбоев: автоматическое восстановление без потери данных транзакции. В случае аппаратных или программных сбоев система должна возвращаться к некоторому согласованному состоянию данных.

Вспомогательные средства должны позволять разработчику или администратору базы данных предсказать и оптимизировать производительность системы.

Автоматическая реорганизация и перемещение: система должна обеспечивать возможность перемещения данных или автоматическую реорганизацию физической структуры.

Типология баз данных

Классификация баз данных может быть произведена по разным признакам, среди которых выделяют следующие.

По *форме представляемой информации* можно выделить фактографические, документальные, мультимедийные, в той или иной степени соответствующие цифровой, символьной и другим (нецифровой и несимвольной) формам представления информации в вычислительной среде.

К последним можно отнести картографические, видео-, аудио-, графические и другие БД.

По *типу хранимой (не мультимедийной) информации* можно выделить фактографические, документальные, лексикографические БД. Лексикографические базы — это классификаторы, кодификаторы, словари основ слов, тезаурусы, рубрикаторы и т. д., которые обычно используются в качестве справочных совместно с документальными или фактографическими БД. Документальные базы подразделяются по уровню представления информации на полнотекстовые (так называемые «первичные» документы) и библиографическо-реферативные («вторичные» документы, отражающие на адресном и содержательном уровнях первичный документ).

По *типу используемой модели данных* выделяют три классических класса БД: иерархические, сетевые, реляционные. Развитие технологий обработки данных привело к появлению постреляционных, объектно-ориентированных, многомерных БД, которые в той или иной степени соответствуют трем упомянутым классическим моделям.

По *топологии хранения* данных различают локальные и распределенные БД.

По *типологии доступа и характеру использования* хранимой информации БД могут быть разделены на специализированные и интегрированные.

По *функциональному назначению* (характеру решаемых с помощью БД задач и, соответственно, характеру использования данных) можно выделить операционные и справочно-информационные. К последним можно отнести ретроспективные БД (электронные каталоги библиотек, БД статистической информации и т. д.), которые используются для информационной поддержки основной деятельности и не предполагают внесения изменений в уже существующие записи, например, по результатам этой деятельности. Операционные БД предназначены для управления различными технологическими процессами. В этом случае данные не только извлекаются из БД, но и изменяются (добавляются) в том числе в результате этого использования.

По *сфере возможного применения* можно различать универсальные и специализированные (или проблемно-ориентированные) системы.

По *степени доступности* можно выделить общедоступные и БД с ограниченным доступом пользователей. В последнем случае говорят об управляемом доступе, индивидуально определяющем не только набор доступных данных, но и характер операций, которые доступны пользователю.

Следует отметить, что представленная классификация не является полной и исчерпывающей. Она в большей степени отражает исторически сложившееся состояние дел в сфере деятельности, связанной с разработкой и применением баз данных.

С другой стороны, БД могут соотноситься с различными уровнями *информационных процессов:* уровень информационных технологий (ИТ), уровень системы (ИС), уровень информационных ресурсов (ИР).

На уровне информационных технологий БД определяется как взаимосвязанная совокупность файлов ОС, содержащих данные о предметной области решаемой задачи. При этом основное внимание уделяется *физической структуре БД.*

На уровне информационных систем БД рассматривается как компонент, представляющий собой информационную модель предметной области. Здесь наиболее важной является проблема *логической структуры БД.*

При рассмотрении на уровне информационных ресурсов БД трактуется как элемент мировых ИР. Основной характеристикой здесь является *содержание БД,* хотя и структуры данных также немаловажны.

Основное внимание в данном пособии будет уделяться рассмотрению БД на уровне технологии и систем, уровень информационных ресурсов будет вкратце рассмотрен только в настоящей главе.

Программные средства баз данных. *Оболочки информационных систем* (системы программирования ИС) представляют собой гибкие программные комплексы, настраиваемые на задачи пользователя. Наиболее распространенными классами данных программных средств являются *системы управления базами данных* (СУБД) и оболочки *автоматизированных информационно-поисковых систем* (АИПС).

Информационно-поисковые системы. Под АИПС принято понимать открытый (обычно) или замкнутый (реже) программный продукт, предназначенный для реализации практически большинства функций (процессов) — *ввод, обработка, хранение, поиск, представление данных* (организованных в записи или документы, находящиеся в БД). В этом смысле часто отождествляют АИПС с АИС, и это трудно оспаривать.

Среди АИПС принято выделять:

• *фактографические системы* (отличающиеся фиксированной структурой данных или записей), для разработки которых как правило используются СУБД, поддерживающие *табличные (реляционные) БД;*

• *документальные системы* (отличающиеся неопределенной или переменной структурой данных или документов), для разработки которых часто (но не обязательно) применяют оболочки АИПС.

Системы управления базами данных и программирования АИС. Среди различных программных средств данного класса следует различать три типа:

• СУБД в «чистом виде»;

• СУБД с элементами систем программирования АИС;

• системы программирования АИС с элементами СУБД.

Первый тип фактически относится к начальному этапу развития систем второго (реже — третьего) типов. В этом случае СУБД состоит только из системы интерпретации вызовов (обращений) из пользовательской программы (call-interface) на выборку (корректировку, занесение)

информации из/в БД, причем программа написана на одном из универсальных языков программирования (ЯП), таких как Кобол, Фортран, Паскаль и пр., получивших название *включающие языки СУБД.* Данная система в последующих СУБД (второй тип) получила наименование *ядра.* Соглашения о форматах и структурах такого взаимодействия обычно пытаются оформить в виде некоторого формального *языка (языка ядра)*. В частности, вдохновленная успехами в разработке и распространении универсального ЯП PL/1 (Programming Language #1), фирма IBM разработала описание форматов интерфейса пользовательских программ с БД IMS в форме языка DL/1 (Data Language #1), который, однако, значительного успеха не имел.

Второй тип представляет собой расширение первого в направлении создания универсальной системы разработчика АИС, включающей также специализированные языковые средства. В этом случае СУБД представляет собой совокупность специализированных программных средств, вспомогательных файлов и управляющих таблиц (иногда находящихся в составе БД, реже это файлы ОС), которая обеспечивает доступ пользователей к БД при соблюдении следующих существенных критериев:

• целостность и непротиворечивость данных, описывающих различные аспекты объектов реального мира, защита информации от несанкционированного доступа к чтению/обновлению содержимого БД;

• установление и поддержание связей между зависимыми данными;

• удобство использования данных.

Третий тип представляют собой (разработанные обычно для ПК) системы, содержащие элементы как непроцедурного (язык запросов), так и процедурного (язык программирования) типов во входном языке, предназначенном для управления данными и обработки информации. Элементы СУБД здесь также заключаются в наличии простейшего словаря данных, возможности создания модели предметной области в форме совокупности таблиц, связанных между собой простейшим образом, а также в наличии средств генерации отчетов и управления доступом пользователей.

Семантика баз данных

Как уже отмечалось, база данных не может рассматриваться в отрыве от назначения и особенностей ее использования для решения практических задач, причем обязательно в составе более крупных информационных или технологических автоматизированных систем. Задачи таких систем — не только планирование и управление предприятием, но и интеграция разработки и сопровождения основных и технологических объектов и процессов, диагностика, мониторинг, моделирование. Соответственно, задачи и назначение БД как системы, хранящей информацию обо всех этих составляющих, — обеспечить информационную поддержку этих процессов.

База данных — это отражение реальной предметной области, «действующая» информационная модель, которая, обеспечивая субъект информацией для принятия решения, позволяет в том числе и управлять

объектами и процессами в отражаемой предметной области (ПрО). Такая функциональная направленность (естественно, предполагающая достижение эффективности в первую очередь за счет использования именно БД) обусловливает и обратную зависимость: объекты, процессы и события ПрО выделяются таким образом, чтобы было возможно их представление в виде системы взаимосвязанных данных и процессов, удобных для их последующей (человеко-машинной) обработки.

В каком-то смысле базу данных можно сравнить с сообщением о состоянии предметной области, воспринимаемым некоторым субъектом, задачей которого и является преобразование объектов этой ПрО, причем в своей деятельности субъект руководствуется информацией, извлекаемой именно из этого «сообщения». Схема этого соотношения, приведенная на рисунке 4.1, иллюстрирует еще и то, что система, преобразующая объект, принципиально является комплексной (состоящей, по крайней мере, из двух компонент, работающих с объектами разной природы: субъект преобразования взаимодействует преимущественно с материальными объектами, а БД — с информационными).

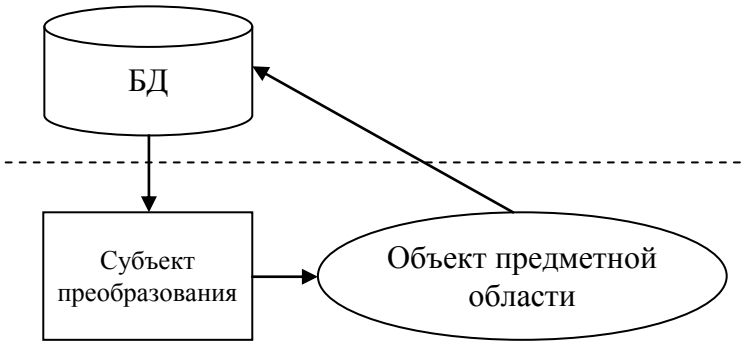


Рисунок 4.1 – Информационная модель преобразования

Для многокомпонентных систем с многоуровневым представлением семантики эффективность обработки достигается через специализированность представления объектов или процессов (а для вычислительных систем — как среды хранения информации — с *единственно возможной* двоичной формой представления) и, в первую очередь, путем сведения представления множества обрабатываемых (локально) объектов к однородности природы и формы их представления. Поэтому, в общем случае для реализации эффективного межуровневого взаимодействия (на каждом из уровней объекты представлены в виде, наиболее адекватном функциональным средствам этого уровня) любая величина должна быть преобразована в соответствии с «контекстом» этого уровня для получения такого ее представления, которое будет значимо для воспринимающего уровня, т. е. может быть обработано средствами этого уровня.

Здесь «контекст» — это декларативное или иногда процедурное определение способа использования элементарных составляющих величины

для получения значения. Например, порядок использования байтов при преобразовании вещественного числа, представленного в двоичной форме, в символьный формат.

Соотношение понятий «величина», «контекст» и «значение» приведено на рисунке 4.2. Здесь значение, получаемое на уровне 1, на следующем рассматривается в свою очередь как величина, которая будет интерпретироваться в соответствии с контекстом своего уровня. Соотношение понятий «величина» и «значение» аналогично соотношению понятий «данные» и «информация». Информация — это значимые для приемника данные, например изменяющие его внутреннее состояние.

Таким образом, можно сказать, что значение в общем случае определяется парой <*контекст, величина*>. Причем, поскольку *контекст* и *величина* имеют разную природу, они должны быть представлены в вычислительной среде самостоятельными, скорее всего, разнотипными объектами.

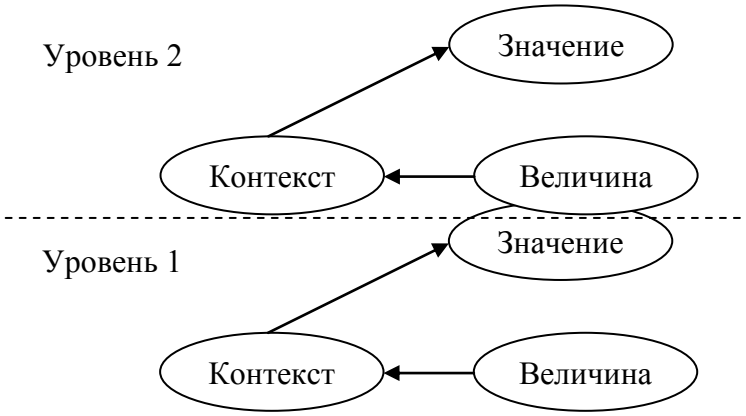


Рисунок 4.2 – Соотношение понятий «величина», «контекст» и «значение»

Такое, хотя и упрощенное представление о БД как о средстве информационных коммуникаций позволяет тем не менее увидеть взаимосвязь вида информации (способа реализации смысла) с формой ее представления и особенностью ее использования.

В этом смысле (с точки зрения способа представления и, соответственно, восприятия) в отдельный класс можно выделить *фактографическую информацию:* такое представление реально существующих событий и явлений, когда они могут быть описаны как *факты,* задаваемые парой <*имя, значение*>, где *имя* — знак, уникально определяющий (идентифицирующий) факт в заданной предметной области, и обычно не нуждающийся в явном определении или доопределении его существа; а *значение* — характеристика, задающая одно из множества возможных состояний.

Таким образом, здесь факт (его значение) задается величиной, например, числовой для параметров, измеримых физически, в том числе и логическими величинами «истина»/«ложь» для указания, свершилось событие или нет.

Можно сказать, что особенностью фактографической информации является практическая очевидность (минимальная неопределенность, не требующая использования сложных или нечетких процедур) идентификации и интерпретации «факта», как его имени, так и состояния. Таким образом, контекст в этом случае в достаточной степени определяется однозначно понимаемым объявлением о назначении базы данных и таким именованием полей данных, когда в качестве имени используется общепринятое, не зависящее от прикладных задач, *имя свойства* (и таким образом определяются характеристические признаки). Такая ситуация предопределяет для пользователя возможность адекватного восприятия содержания: способ интерпретации данных в этом случае практически не может быть неоднозначным, причем для пользователя *определение способа* происходит *неявно* (не требует от него явных действий для определения и использования контекста). Это, с одной стороны, позволяет свести представление предметной области к точной теоретико-множественной модели, а с другой — обусловливает возможность непосредственного использования данных в задачах обработки (на уровне прикладных программ) для генерации новой информации без участия субъекта (человека), внешнего по отношению к машинной среде, обеспечивающего определение и использование контекста. Например, OLAP-технологии[1] баз данных, позволяющие строить на основе множества данных, количественно характеризующих состояние объектов предметной области и представленных обычно регулярными таблицами, новые значения, отражающие это состояние на ином *качественном* уровне, например, интегральные показатели, диаграммы, графики и т. д.

Однако большинство задач, решаемых человеком, не может быть сведено к «фактографическому» представлению и описывается (и, соответственно, представляется в машинной среде) средствами естественного или специализированного языков, оперирующих *лингвистическими переменными,* значение которых может зависеть не только от контекста предметной области, но также и от контекста ближайшего окружения — значения соседних переменных. Причем, появление нового смысла (факта) не обязательно приводит к появлению новой переменной: новый факт представляется с помощью уже существующих переменных. Например, словесные определения философских или географических понятий.

В отличие от ранее рассмотренного фактографического представления, для вербальной формы представления факта (выражениями языка с использованием лингвистических переменных) характерно то, что для задания *имени, значения* и *контекста* может использоваться единый способ

[1] **OLAP** (англ. *online analytical processing*, аналитическая обработка в реальном времени) — технология обработки данных, заключающаяся в подготовке суммарной (агрегированной) информации на основе больших массивов данных, структурированных по многомерному принципу.

и средства — лингвистические переменные одного и того же языка. Например, описание весовых свойств может быть представлено несколькими, но имеющими один смысл, вариантами предложений: «Чугунная заготовка весом 29 килограммов» или «Чугунная заготовка имеет свойство m = 29, где m — вес в килограммах».

Автоматическое приведение такого рода представлений к очевидно наилучшей для этого случая табличной форме, потребовало бы применения трудно реализуемых процедур морфологического и семантического анализов. Однако с другой стороны, выделение смысла (и генерация новой информации) обычно производится человеком, сознание которого (как среда преобразования) ориентировано именно на обработку лингвистических переменных.

Рассматривая процесс автоматизированной генерации новой информации (рисунок 4.3), где в качестве источника исходных данных используются БД, нужно сказать, что отбор и обработка должны быть выделены в отдельные процессы, так как с точки зрения общей (суммарной) эффективности один из них (обычно поиск) должен быть опосредованным — оценка полезности найденной информации производится обычно человеком, так как сознание человека — внешняя по отношению к машине среда, работает со слабоструктурированной информацией эффективнее машин.

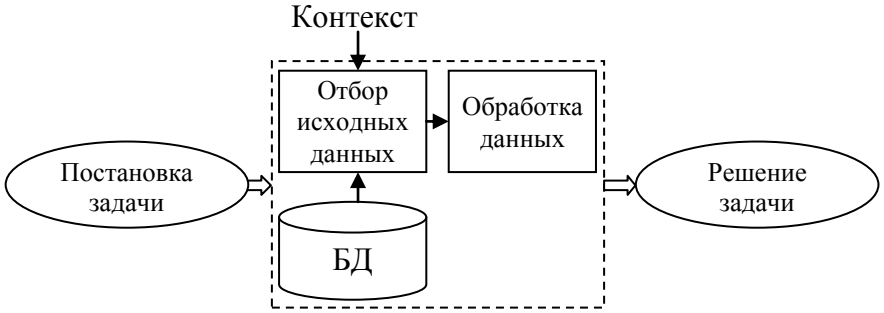


Рисунок 4.3 – Схема процесса автоматизированного решения задач

Случаи, когда информация представляется в форме, не адекватной архитектуре фон-неймановских машин, могут быть обусловлены разными факторами. Рассмотрим следующие случаи.

1.Хорошо структурированная информация, представляемая в графическом или специальном формате. Например, структурные химические формулы, конструкторская документация и т. д. В этом случае для автоматической обработки требуются узкоспециализированные средства, что приводит к общей неунифицированности представления семантических элементов (например, графических примитивов) на уровне данных.

2.Информация, точная по содержанию, но вариантно представляемая по форме. Например, описание в текстовом виде численно задаваемых параметров изделия. Лингвистические переменные в этом случае имеют точное значение, однако построение универсальной процедуры автоматического выделения факта из текста трудоемко и потому

нецелесообразно.

3.Слабоструктурированная информация, обычно представляемая в текстовой форме. Например, учебная или научная публикация, где новые понятия строятся на основании ранее определенных. В этом случае лингвистические переменные могут принимать новые, ранее не определенные значения, которые определяются контекстом — ближним (словосочетания) или общим (темой сообщения).

Возвращаясь к процедуре поиска как важнейшей составляющей использования баз данных, еще раз отметим, что критерий отбора должен содержать не только величину (например, слово), но и контекст.

В реальных системах поиск документальной информации, представленной в текстовой форме, производится по вторичным документам — специально создаваемым поисковым образам, точно идентифицирующим сам документ как единицу хранения, и приблизительно, в краткой форме, путем *перечисления* основных понятий, отражающий смысловое содержание. Такой подход позволяет построить процедуры поиска на основе теоретико-множественной модели с точной логикой отбора по критерию наличия заданного сочетания терминов запроса в списке терминов поискового образа. Однако контекст использования терминов должен быть доопределен отдельно — либо во время поиска, например, указанием тематической области, либо после отбора из базы — во время ознакомления человека с содержанием найденного.

Определение контекста предметной области в целом осуществляется с помощью тезаурусов терминологических систем, фиксирующих с помощью родо-видовых и других отношений роль и семантику дескрипторов — выделенных терминов, которые используются для формирования поисковых образов документов.

Для доопределения смысла термина в составе поискового образа документа в первых поколениях автоматизированных информационных систем применялись специальные указатели роли, однако их использование было трудоемко и требовало специальной подготовки пользователя, поэтому в современных системах не применяется.

Другой важный фактор, влияющий на эффективность работы человека с информацией — это форма хранения и представления — структура и оформление документа. Это особенно заметно при работе с объемными полнотекстовыми документами, причем иногда определяется на уровне машинного формата (например, DOC, PDF, HTML и т. д.), от выбора которого зависит возможность дальнейшей обработки.

В том случае когда для хранения информации используются базы данных, структура документов может быть определена двумя путями:

• так же как и для фактографических БД, заданием схемы — последовательности именованных типизированных полей данных;

• контекстным определением — использованием специализированных языков разметки (например, HTML или XML), задающим индивидуальные особенности представления материала каждого документа.

Использование встраиваемых определений структуры позволяет ввести «самоопределяемые» форматы представления документов. Это обеспечивает практически неограниченную гибкость при организации хранения коллекций разнородных документов, однако создает семантические проблемы согласованного использования материала (из-за возможности различной интерпретации определений), что в свою очередь требует создания доступного всем пользователям репозитария метаинформации — описаний природы и способов представления информации.

Введение в технологии машинной обработки данных и основные определения

Реальные базы данных промышленного масштаба содержат миллионы записей, данные которых описывают состояния и взаимосвязи многих и многих объектов реального мира. Требования, предъявляемые пользователями к автоматизированным или автоматическим системам, обрабатывающим эти данные, обусловливают и требования к параметрам подсистем внешней памяти, в первую очередь, предполагают высокую оперативность доступа.

Важной особенностью здесь является то, что архитектура систем и технологий управления данными непосредственно связана с двумя следующими значительными, хотя и противоположными обстоятельствами:

• непредсказуемой вариантностью представления данных в прикладной программе, зависящей от разнообразных особенностей пользовательских задач;

• жесткостью технических решений устройств внешней памяти, выражающейся в функциональной простоте[2] операций и ограниченности форм представления данных.

Высокая эффективность решений в области обработки данных достигается введением промежуточных слоев специализированных технических и программных средств. Характер проблем и архитектурно-технологические решения такого рода достаточно полно иллюстрируются приведенной на рисунке 4.4 примерной схемой реализации операций ввода-вывода — взаимодействия прикладной программы с компонентами операционной системы и устройствами внешней памяти. Здесь *специализация* компонент выражается в том, что по существу каждый из них реализует различные способы работы с потоком данных (и в частности, его

[2]Требование операционной простоты определяется производственными и экономическими причинами: устройство должно быть надежным в использовании и дешевым в изготовлении (т. е. содержать минимум механических компонент и сложной логики).

Функциональная ограниченность управления данными, кроме того, диктуется еще и требованием *унифицированности:* устройство должно одинаково эффективно и стандартным способом использоваться в составе различных вычислительных и операционных систем, даже если со временем отдельные компоненты систем будут принципиально меняться.

фрагментацию на блоки), что и обеспечивает, с одной стороны, необходимый уровень декомпозиции и идентификации логических/физических записей, а с другой — независимость физического и логического уровней представления данных.

Здесь термины *логический* и *физический* отражают различия аспектов представления данных. *Логическое представление* указывает на то, как данные используются в прикладной программе, т. е. отражают логику обработки. *Физическое представление* — это то, как данное хранятся на *физическом носителе.*

Рисунок 4.4 – Примерная схема организации ввода-вывода

Будем считать *логической записью* идентифицируемую (именованную) *совокупность элементов или агрегатов данных,* воспринимаемую прикладной программой как единое целое при обмене информацией с внешней памятью (по крайней мере, для операций ввода-вывода).

Физической записью будем считать совокупность данных, которая может быть считана или записана как единое целое *одной командой ввода-вывода.* Важно, что для компонент различного уровня в технологической цепи ввода-вывода состав и структура физической записи может быть разной.

Структура данных и их взаимосвязь в случаях логического и физического представления могут не совпадать. Например, а) одна физическая запись может включать несколько логических; б) порядок следования элементов данных в физической записи может быть изменен для

оптимизации использования пространства памяти. То есть, если логическая структура может варьироваться в широком диапазоне и даже представляться, например, вариантными записями, то физическая — практически всегда представлена жесткой структурой, причем в значительной степени определяемой типом носителя.

Типы, форматы, структуры данных

Структура информационных единиц, обрабатываемых на ЭВМ, определяется следующими понятиями:

• *тип данных,* или совокупность соглашений о программно-аппаратурной форме представления и обработки, а также ввода, контроля и вывода элементарных данных;

• *структуры данных* — способы композиции простых данных в агрегаты и операции над ними;

•*форматы файлов* — представление информации им уровне взаимодействия операционной системы с прикладными программами.

Типы данных. Ранние языки программирования — Фортран, Алгол — были ориентированы исключительно на вычисления и не содержали систем типов и структур данных. Типы числовых данных Алгола: INTEGER (целое число), REAL (действительное) — различаются диапазонами изменения, внутренними представлениями и применяемыми командами процессора ЭВМ (соответственно арифметика с фиксированной и плавающей точкой). Нечисловые данные представлены типом BOOLEAN — логические, имеющие диапазон значений {TRUE, FALSE}.

Появившиеся позже языки программирования COBOL, PL/1, Pascal уже предусматривают новые типы данных:

• символьные (цифры, буквы, знаки препинания и пр.);

• числовые символьные для вывода;

• числовые двоичные для вычислений;

• числовые десятичные (цифры 0—9) для вывода и вычислений.

Структуры данных. В языке программирования Алгол были определены два типа структур: *элементарные данные и массивы* (векторы, матрицы, тензоры, состоящие из арифметических или логических переменных). Основным нововведением, появившимся первоначально в Коболе, (затем в PL/1, Паскале и пр.) являются *агрегаты данных* (структуры, записи), представляющие собой именованные комплексы переменных разного типа, описывающих некоторый объект или образующих некоторый достаточно сложный документ.

Термин *запись* подразумевает наличие множества аналогичных по структуре агрегатов, образующих *файл* (картотеку), содержащих данные по совокупности однородных объектов. Элементы данных образуют поля, среди которых выделяются элементарные и групповые (агрегатные).

Появление СУБД и АИПС приводит к появлению новых разновидностей структур:

• множественные поля данных;

• периодические групповые поля;

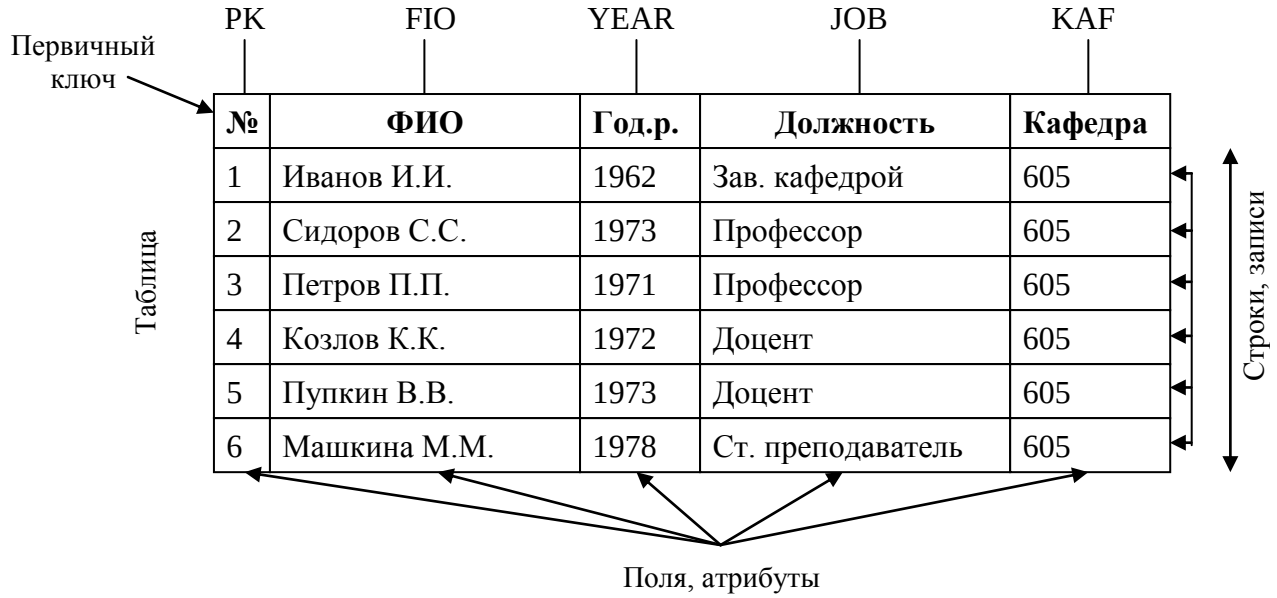
• текстовые объекты (документы), имеющие иерархическую структуру (документ, сегмент, предложение, слово).

Форматы файлов. В зависимости от типа и назначения файлов и возможностей ОС (методов доступа) файл может передаваться в прикладную программу как целое или блоками (физическими записями) либо логическими записями (строками, словами, символами).

Например, в системе OS/360 основную роль играли два типа файлов: символьные (исходные программы или данные) и двоичные (программы в машинных кодах). В современных системах активно используется значительно большее разнообразие файлов, например, *текстовые файлы* — обобщенное название для простых и размеченных текстов, ASCII-файлов и других наборов данных символьной информации.

Реляционная модель данных

Реляционная[3] модель является удобной и наиболее привычной формой представления данных в виде таблицы. В отличие от иерархической и сетевой моделей, такой способ представления: 1) понятен пользователю-непрограммисту; 2) позволяет легко изменять схему — присоединять новые элементы данных и записи без изменения соответствующих подсхем; 3) обеспечивает необходимую гибкость при обработке непредвиденных запросов. К тому же любая сетевая или иерархическая схема может быть представлена двумерными отношениями.



№	ФИО	Год.р.	Должность	Кафедра
1	Иванов И.И.	1962	Зав. кафедрой	605
2	Сидоров С.С.	1973	Профессор	605
3	Петров П.П.	1971	Профессор	605
4	Козлов К.К.	1972	Доцент	605
5	Пупкин В.В.	1973	Доцент	605
6	Машкина М.М.	1978	Ст. преподаватель	605

Рисунок 4.5 – Основные понятия реляционной модели

[3] В математических дисциплинах понятию «таблица» соответствует понятие «отношение» (relation). Отсюда и произошло название модели – реляционная. То есть, применительно к базам данных понятия «реляционная БД» и «табличная БД» по существу являются синонимами.

Одним из основных преимуществ реляционной модели является ее однородность. Все данные рассматриваются как хранимые в таблицах, в которых каждая строка имеет один и тот же формат. Каждая строка в таблице представляет некоторый объект реального мира или соотношение между объектами. Пользователь модели сам должен для себя решить вопрос, обладают ли соответствующие сущности реального мира однородностью. Этим самым решается проблема пригодности модели для предполагаемого применения.

Первичный ключ — это столбец или некоторое подмножество столбцов, которые уникально, т. е. единственным образом определяют строки. Первичный ключ, который включает более одного столбца, называется множественным, или комбинированным, или составным. Правило целостности объектов утверждает, что первичный ключ не может быть полностью или частично пустым, т. е. иметь значение null.Остальные ключи, которые можно также использовать в качестве первичных, называются потенциальными или *альтернативными* ключами.

Внешний ключ — это столбец или подмножество одной таблицы, который может служить в качестве первичного ключа для другой таблицы. *Внешний ключ* таблицы является ссылкой на первичный ключ другой таблицы. Правило ссылочной целостности гласит, что внешний ключ может быть либо пустым, либо соответствовать значению первичного ключа, на который он ссылается. Внешние ключи являются неотъемлемой частью реляционной модели, поскольку реализуют связи между таблицами базы данных.

Внешний ключ, как и первичный ключ, тоже может представлять собой комбинацию столбцов. На практике внешний ключ всегда будет составным (состоящим из нескольких столбцов), если он ссылается на составной первичный ключ в другой таблице. Очевидно, что количество столбцов и их типы данных в первичном и внешнем ключах совпадают.

Если таблица связана с несколькими другими таблицами, она может иметь несколько внешних ключей.

Модель предъявляет к таблицам следующие требования:

1)данные в ячейках таблицы должны быть структурно неделимыми[4];

2)данные в одном столбце должны быть одного типа;

3)каждый столбец должен быть уникальным (недопустимо дублирование столбцов);

4)столбцы размещаются в произвольном порядке;

5)строки размещаются и таблице также в произвольном порядке;

6)столбцы имеют уникальные наименования.

[4] Недопустимо, чтобы в ячейке таблицы содержались более одной порции информации, что иногда происходит при создании композиционных данных. Примером служит идентификационный номер автомобиля. Если записать его в одну ячейку, то будет нарушен принцип неделимости информации, поскольку в ячейке окажутся разделяемые данные, имеющие разную информационную сущность, такие как наименование модели, номер кузова, двигателя, сведения о предприятии-изготовителе и т. д.

4.4 Базовые сведения о компьютерной графике и геометрии

Компьютерная графика – это раздел информатики, в котором изучаются методы и средства для преобразования данных в графическую форму представления и обратно с помощью ЭВМ. В компьютерной графике рассматриваются следующие задачи:

- представление изображения в компьютерной графике;
- подготовка изображения к визуализации;
- создание изображения;
- осуществление действий с изображением.

Под компьютерной графикой обычно понимают автоматизацию процессов подготовки, преобразования, хранения и воспроизведения графической информации с помощью компьютера. Под графической информацией понимаются модели объектов и их изображения.

Основные понятия компьютерной графики

Разрешение. В компьютерной графике с понятием разрешения обычно происходит больше всего путаницы, поскольку приходится иметь дело сразу с несколькими свойствами разных объектов. Следует четко различать: разрешение экрана, разрешение печатающего устройства и разрешение изображения. Все эти понятия относятся к разным объектам. Друг с другом эти виды разрешения никак не связаны пока не потребуется узнать, какой физический размер будет иметь картинка на экране монитора, отпечаток на бумаге или файл на жестком диске.

Разрешение экрана - это свойство компьютерной системы (зависит от монитора и видеокарты) и операционной системы (зависит от настроек). Разрешение экрана измеряется в *пикселях* (точках) и определяет размер изображения, которое может поместиться на экране целиком.

Разрешение принтера - это свойство принтера, выражающее количество отдельных точек, которые могут быть напечатаны на участке единичной длины. Оно измеряется в единицах *dpi* (dots per inch, точки на дюйм) и определяет размер изображения при заданном качестве или, наоборот, качество изображения при заданном размере.

Разрешение изображения - это свойство самого изображения. Оно тоже измеряется в точках на дюйм и задается при создании или редактировании изображения. Так, для просмотра изображения на экране достаточно, чтобы оно имело разрешение *72 dpi*, а для печати на принтере - не меньше чем *300 dpi*. Значение разрешения изображения хранится в файле изображения.

Физический размер изображения определяет размер рисунка по вертикали (высота) и горизонтали (ширина) может измеряться как в пикселях, так и в единицах длины (миллиметрах, сантиметрах, дюймах). Он задается при создании (редактировании) изображения и хранится вместе с файлом. Если изображение готовят для демонстрации на экране, то его ширину и высоту задают в пикселях, чтобы знать, какую часть экрана оно занимает. Если изображение готовят для печати, то его размер удобнее

указывать в единицах длины, чтобы знать, какую часть листа бумаги оно займет.

Физический размер и разрешение изображения неразрывно связаны друг с другом. При изменении разрешения автоматически меняется физический размер.

При работе с цветом используются понятия *глубины цвета* (его еще называют цветовым разрешением) и *цветовой модели*.

Для кодирования цвета пикселя изображения может быть выделено разное количество бит. От этого зависит, сколько цветов на экране может отображаться одновременно. Чем больше длина двоичного кода цвета, тем больше цветов можно использовать в рисунке. *Глубина цвета* - это количество бит, которое используют для кодирования цвета одного пикселя. Для кодирования двухцветного (черно-белого) изображения достаточно выделить по одному биту на представление цвета каждого пикселя. Выделение одного байта (2^8) позволяет закодировать 256 различных цветовых оттенков. Два байта (16 битов) позволяют определить 65536 различных цветов. Этот режим называется High Color. Если для кодирования цвета используются три байта (24 бита), возможно одновременное отображение 16,5 миллионов цветов. Этот режим называется True Color. От глубины цвета зависит размер файла, в котором сохранено изображение.

Цвета в природе редко являются простыми. Большинство цветовых оттенков образуется смешением основных цветов. Способ разделения цветового оттенка на составляющие компоненты называется *цветовой моделью*. С практической точки зрения цветовому разрешению монитора близко понятие цветового охвата. Под ним подразумевается диапазон цветов, который можно воспроизвести с помощью того или иного устройства вывода (монитор, принтер и т.д.). В соответствии с принципами формирования изображения разработаны способы разделения цветового оттенка на составляющие компоненты, называемые цветовыми моделями. Существует много различных типов цветовых моделей, но в компьютерной графике, как правило, чаще всего применяются три: RGB, CMYK, HSB.

Процесс получения различных цветов с помощью нескольких основных (первичных) излучений или красок называется цветовым синтезом. Существует два принципиально различных метода цветового синтеза: *аддитивный* (англ. add - сложение) и *субтрактивный* (англ. subtract - вычитание) синтезы.

Аддитивный цвет получается при соединении *света* разных цветов. В этой схеме отсутствие всех цветов представляет собой чёрный цвет, а присутствие всех цветов - белый. *Схема аддитивных цветов работает с излучаемым светом*.

В схеме субтрактивных цветов происходит обратный процесс. Здесь получается какой-либо цвет при вычитании других цветов из общего луча света. В этой схеме белый цвет появляется в результате отсутствия всех

цветов, тогда как их присутствие даёт чёрный цвет. *Схема субтрактивных цветов работает с отражённым светом*.

RGB (аббревиатура английских слов **R**ed, **G**reen, **B**lue — красный, зелёный, синий) — аддитивная цветовая модель, как правило, описывающая способ синтеза цвета для цветовоспроизведения.

Выбор основных цветов обусловлен особенностями физиологии восприятия цвета сетчаткой человеческого глаза. Цветовая модель RGB нашла широкое применение в технике.

Изображение в данной цветовой модели состоит из трёх каналов. При смешении основных цветов (основными цветами считаются красный, зелёный и синий) — например, синего (B) и красного (R), мы получаем пурпурный (M - magenta), при смешении зеленого (G) и красного (R) — жёлтый (Y - yellow), при смешении зеленого (G) и синего (B) — голубой (C - cyan, иногда говорят "цианový"). При смешении всех трёх цветовых компонентов мы получаем белый цвет (W).

Цветовая модель RGB имеет по многим тонам цвета более широкий цветовой охват (может представить более насыщенные цвета), чем типичный охват цветов CMYK, поэтому иногда изображения, замечательно выглядящие в RGB, значительно тускнеют и гаснут в CMYK.

Цветовые модели расположены в трехмерной системе координат, образующей цветовое пространство, так как из законов Гроссмана следует, что цвет можно выразить точкой в трехмерном пространстве.

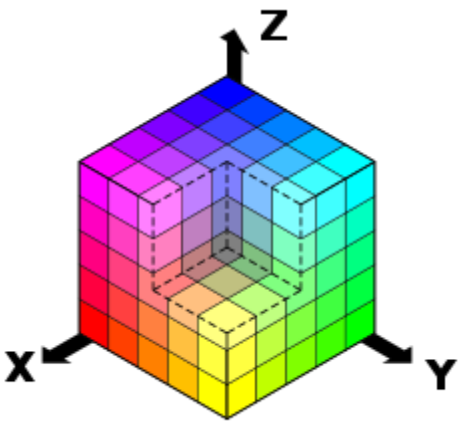
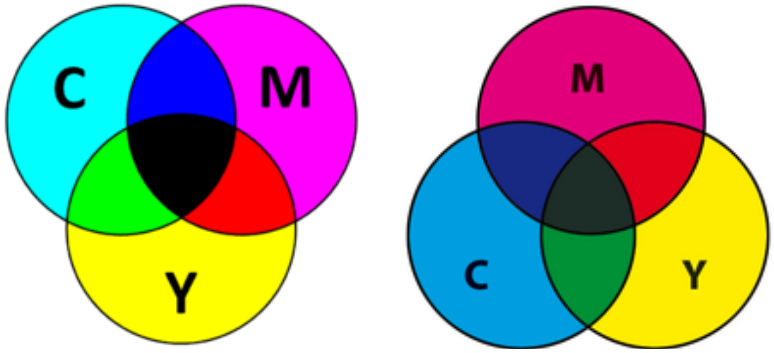


Рисунок 4.6 – RGB-цветовая модель, представленная в виде куба

Четырёхцветная автотипия (CMYK: Cyan, Magenta, Yellow, Key color) — субтрактивная схема формирования цвета, используемая прежде всего в полиграфии для стандартной триадной печати (рисунок 4.7). Схема CMYK, как правило, обладает (сравнительно с RGB) небольшим цветовым охватом.

По-русски эти цвета часто называют голубым, пурпурным и жёлтым, хотя первый точнее называть сине-зелёным, а маджента — лишь часть пурпурного спектра. Печать четырьмя красками, соответствующими CMYK, также называют печатью триадными красками.

Цвет в CMYK зависит не только от спектральных характеристик красителей и от способа их нанесения, но и их количества, характеристик бумаги и других факторов. Фактически, цифры CMYK являются лишь набором аппаратных данных для фотонаборного автомата или CTP (Computer to Plate - технология изготовления печатных форм в полиграфии) и не определяют цвет однозначно (рисунок 4.7).



а) б)

Рисунок 4.7 – Синтез цветов по схеме CMYK

а) Схема субтрактивного синтеза в CMYK

б) Наложение реальных типографских красок CMY

В CMYK используются четыре цвета, первые три в аббревиатуре названы по первой букве цвета, а в качестве четвёртого используется чёрный. Одна из версий утверждает, что K — сокращение от английского *blacK*.

Так как модель CMYK применяют в основном в полиграфии при цветной печати, а бумага и прочие печатные материалы являются поверхностями, отражающими свет, удобнее считать, какое количество света отразилось от той или иной поверхности, нежели сколько поглотилось.

Несмотря на то, что чёрный цвет можно получать смешением в равной пропорции пурпурного, голубого и жёлтого красителей, по ряду причин (чистота цвета, переувлажнение бумаги и др.) такой подход обычно неудовлетворителен. Основные причины использования дополнительного чёрного пигмента таковы:

– На практике в силу неидеальности красителей и погрешностей в пропорциях компонентов смешение реальных пурпурного, голубого и жёлтого цветов даёт скорее грязно-коричневый или грязно-серый цвет; триадные краски не дают той глубины и насыщенности, которая достигается использованием настоящего чёрного. Так как чистота и насыщенность чёрного цвета, а также стабильность оттенка нейтральных (серых) областей чрезвычайно важны в печатном процессе, был введён ещё один цвет.

– При выводе мелких чёрных деталей изображения или текста без использования чёрного пигмента возрастает риск неприводки (недостаточно точное совпадение точек нанесения) пурпурного, голубого и жёлтого цветов. Увеличение же точности печатающего аппарата требует неадекватных затрат.

– Смешение 100% пурпурного, голубого и жёлтого пигментов в одной точке в случае струйной печати существенно смачивает бумагу, деформирует её и увеличивает время просушки. Аналогичные проблемы с так называемой *суммой красок* возникают и в офсетной печати. В зависимости от типа материала и технологии печати ограничение по сумме красок может быть ниже 300 % (например, в газетной печати типичное ограничение 260—280 %), что делает технически невозможным синтез насыщенного чёрного из трёх стопроцентных компонентов триады.

– Чёрный пигмент (в качестве которого, как правило, используется сажа) существенно дешевле остальных трёх.

HSV (англ. *Hue, Saturation, Value — тон, насыщенность, значение*) или **HSB** (англ. *Hue, Saturation, Brightness — оттенок, насыщенность, яркость*) — цветовая модель, в которой координатами цвета являются:

- **Hue** — цветовой тон, (например, красный, зелёный или сине-голубой). Варьируется в пределах 0—360°, однако иногда приводится к диапазону 0—100 или 0—1.
- **Saturation** — насыщенность. Варьируется в пределах 0—100 или 0—1. Чем больше этот параметр, тем «чище» цвет, поэтому этот параметр иногда называют чистотой цвета. А чем ближе этот параметр к нулю, тем ближе цвет к нейтральному серому.
- **Value** (значение цвета) или Brightness — яркость. Также задаётся в пределах 0—100 и 0—1.

Модель была создана Элви Реем Смитом, одним из основателей Pixar, в 1978 году. Она является нелинейным преобразованием модели RGB.

Цвет, представленный в HSV, зависит от устройства, на которое он будет выведен, так как HSV — преобразование модели RGB, которая тоже зависит от устройства.

Простейший способ отобразить HSV в трёхмерное пространство — воспользоваться цилиндрической системой координат (рисунок 4.8). Здесь координата *H* определяется полярным углом, *S* — радиус-вектором, а *V* — *Z*-координатой. То есть, оттенок изменяется при движении вдоль окружности цилиндра, насыщенность — вдоль радиуса, а яркость — вдоль высоты.

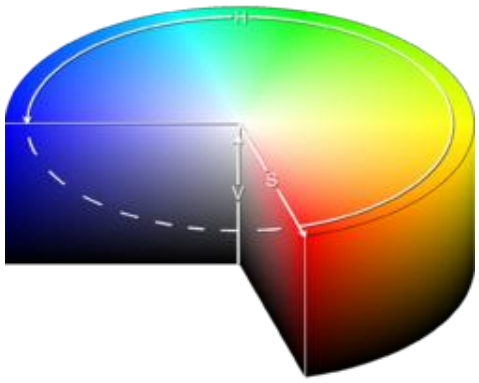
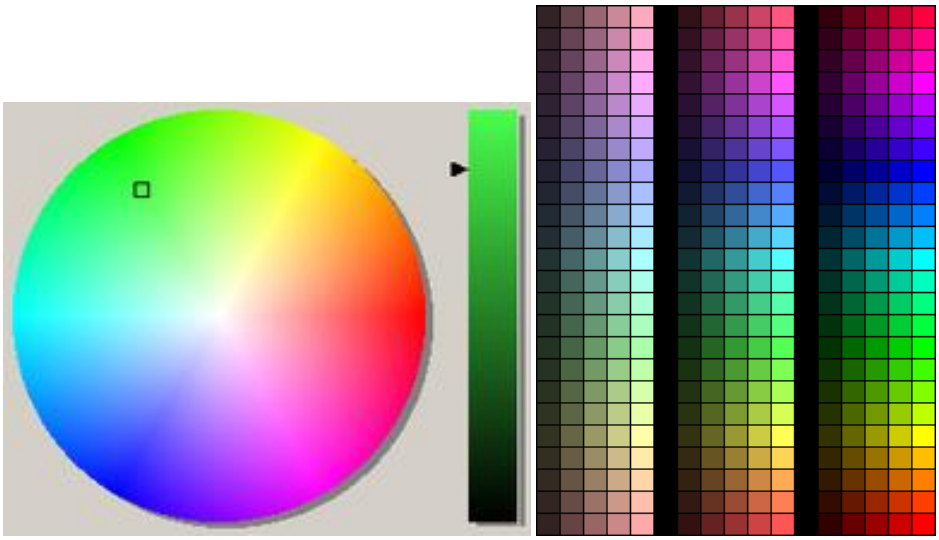


Рисунок 4.8 – Отображение модели HSV в трёхмерное пространство

Модель HSV часто используется в программах компьютерной графики, так как удобна для человека. Ниже представлены способы «разворачивания» трёхмерного пространства HSV на двухмерный экран компьютера.



а) б)

Рисунок 4.9 – Отображение модели HSV на двухмерный экран
а) Цветовой круг; б) Карта цветов

Часто художники предпочитают использовать HSV вместо других моделей, потому что они считают, что устройство HSV ближе к человеческому восприятию цветов. RGB и CMYK определяют цвет как комбинацию основных цветов (красного, зелёного и синего или жёлтого, пурпурного, голубого и чёрного соответственно), в то время как компоненты цвета в HSV отображают информацию о цвете в более привычной человеку форме.

По способам задания изображений графику можно разделить на категории: *двухмерную* и *трехмерную*.

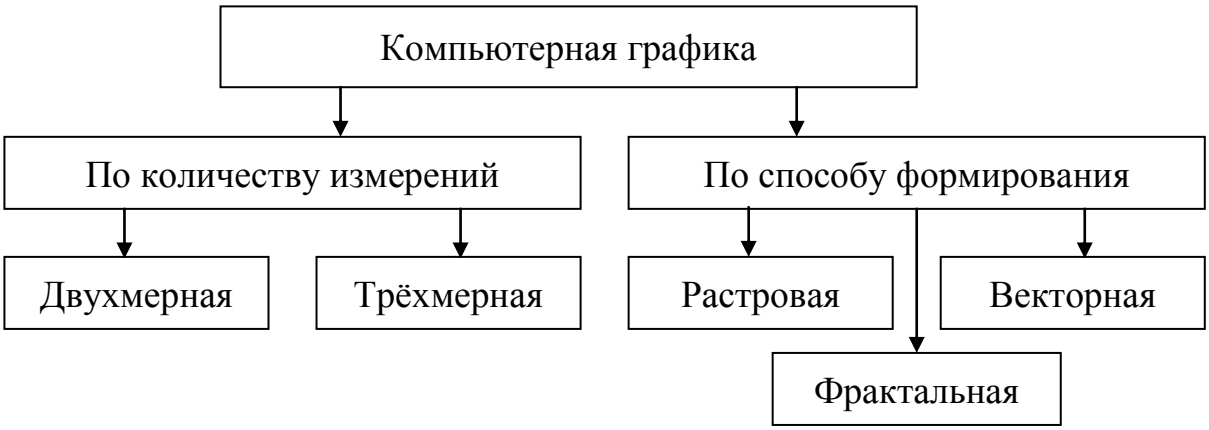


Рисунок 4.10 – Классификация компьютерной графики

Двухмерная (2D — от two dimensions — «два измерения») компьютерная графика классифицируется по типу представления графической информации, и следующими из него алгоритмами обработки изображений. Обычно компьютерную графику разделяют на векторную и растровую, хотя обособляют ещё и фрактальный тип представления изображений.

Растровая графика

Растровое изображение храниться в компьютере в виде массива числовых величин. Предполагается, что этот массив является прямоугольным, с определенным числом строк и столбцов. Каждая числовая величина представляет значение пикселя (pixel – сокращенно от picture elements), записанного в этом массиве. Этот массив в целом называется «пиксельной картой» («pixel map»). Используют, также термин «битовая карта» («bitmap»).

На рисунке 4.11 представлен пример растрового изображения

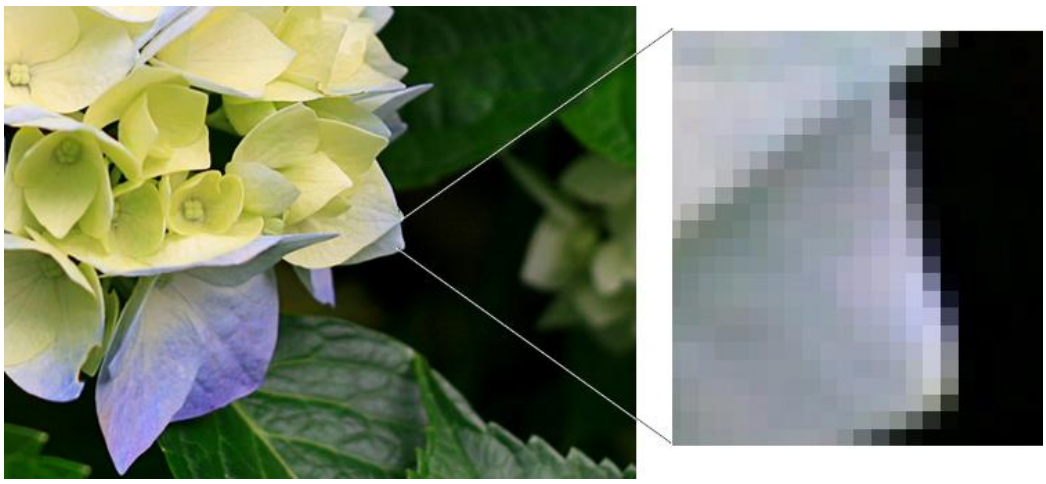


Рисунок 4.11 – Растровое изображение

Важными характеристиками растрового изображения являются:

- размер; может указываться отдельно количество пикселей по ширине и высоте (1024×768, 640×480, …) или же, редко, общее количество пикселей (часто измеряется в мегапикселях);
- количество используемых цветов или глубина цвета;
- цветовое пространство (цветовая модель);
- разрешение.

Достоинства растровой графики:

- Растровая графика позволяет создать (воспроизвести) практически любой рисунок, вне зависимости от сложности, в отличие, например, от векторной, где невозможно точно передать эффект перехода от одного цвета к другому без потерь в размере файла.
- Распространённость — растровая графика используется сейчас практически во всех сферах человеческой деятельности.

- Высокая скорость обработки сложных изображений, если не нужно масштабирование.
- Растровое представление изображения естественно для большинства устройств ввода-вывода графической информации, таких как мониторы (за исключением векторных), матричные и струйные принтеры, цифровые фотоаппараты, сканеры, а также сотовые телефоны.

Недостатки:

- Большой размер файлов у простых изображений.
- Невозможность идеального масштабирования.

Форматы файлов растровых изображений

Одна из важнейших технологий компьютерной графики заключается в хранении файлов растровых изображений, в которых содержится информация, необходимая компьютеру для воссоздания изображения. Наиболее существенна при этом та часть процесса, где происходит преобразование информации в битовую карту. Существует несколько форматов файлов растровой графики, и каждый формат предусматривает собственный способ кодирования информации о пикселях и другой присущей изображениям информации. Наиболее распространенные форматы приведены в таблице 4.1

Таблица 4.1 – Распространенные форматы растровой графики

Формат	**Описание**	**Глубина цвета**	**Кодирование нескольких изображений**
BMP (Bitmap)	Формат хранения растровых изображений.	48	нет
GIF (Graphics Interchange Format)	Формат для обмена изображениями по сети. Используется для получения простейшей анимации	8	есть
JPEG (Joint Photographic Experts Group)	Является широко используемым методом сжатия фотоизображений с потерей качества.	24	нет
PNG (Portable Network Graphics)	Популярный растровый формат хранения графической информации, использующий сжатие без потерь.	48	нет
TIFF (Tagged Image File Format)	Формат хранения растровых графических изображений с большой глубиной цвета, используется при сканировании и распознавании текста.	24	есть

Растровый графический редактор — специализированная программа, предназначенная для создания и обработки растровых изображений. Подобные программные продукты нашли широкое применение в работе художников-иллюстраторов, при подготовке изображений к печати типографским способом или на фотобумаге, публикации в интернете.

Растровые графические редакторы позволяют пользователю рисовать и редактировать изображения на экране компьютера, а также сохранять их в различных растровых форматах.

В противоположность векторным редакторам, растровые используют для представления изображений матрицу окрашенных точек (bit map). Однако, большинство современных растровых редакторов содержат векторные инструменты редактирования в качестве вспомогательных.

В качестве примера растровых графических редакторов можно привести Microsoft Paint, Adobe Photoshop, ACDSee PhotoEditor, Corel Photo-Paint, Gimp.

Векторная графика — способ представления объектов и изображений в компьютерной графике, основанный на использовании элементарных геометрических объектов, таких как точки, линии, сплайны и многоугольники. Объекты векторной графики являются графическими изображениями математических функций. Термин используется в противоположность к растровой графике.

Рисунок 4.12 – Пример векторного изображения

Способ хранения изображения

Рассмотрим, к примеру, такой графический примитив, как окружность радиуса *r*. Для её построения необходимо и достаточно следующих исходных данных:

1. координаты центра окружности;
2. значение радиуса *r*;
3. цвет заполнения (если окружность не прозрачная);
4. цвет и толщина контура (в случае наличия контура).

Данный пример показывает основное достоинство векторной графики – описание объекта является простым и занимает мало памяти. Для описания этой же окружности средствами растровой графики потребовалось бы запомнить каждую отдельную точку изображения, что заняло бы гораздо больше памяти.

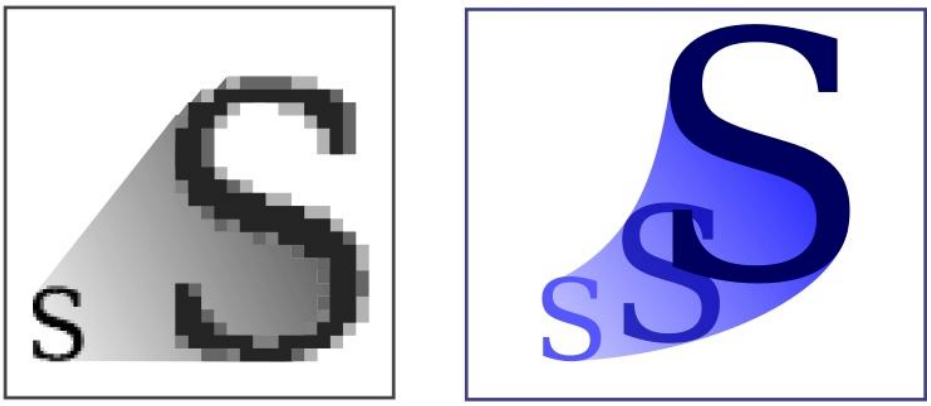
Преимущества векторного способа описания графики над растровой графикой:

– Размер, занимаемый описательной частью, не зависит от реальной величины объекта, что позволяет, используя минимальное количество информации, описать сколько угодно большой объект файлом минимального размера.

– В связи с тем, что информация об объекте хранится в описательной форме, можно бесконечно увеличить графический примитив, например, дугу окружности, и она останется гладкой. С другой стороны, если кривая представлена в виде ломаной линии, увеличение покажет, что она на самом деле не кривая.

– Параметры объектов хранятся и могут быть легко изменены. Также это означает что перемещение, масштабирование, вращение, заполнение и т. д. не ухудшает качества рисунка. Более того, обычно указывают размеры в аппаратно-независимых единицах (англ. device-independent unit), которые ведут к наилучшей возможной растеризации на растровых устройствах.

– При увеличении или уменьшении объектов толщина линий может быть задана постоянной величиной, независимо от реального контура.



а) б)

Рисунок 4.13 – Разница между векторной и растровой графикой при увеличении

а) иллюстрация, увеличенная в 8 раз как растровое изображение;

б) иллюстрация, увеличенная в 8 раз как векторное изображение.

Фундаментальные *недостатки* векторной графики

– Не каждый объект может быть легко изображен в векторном виде — для подобного оригинальному изображению может потребоваться очень большое количество объектов и их сложности, что негативно влияет на

количество памяти, занимаемой изображением, и на время для его отображения (отрисовки).

– Перевод векторной графики в растр достаточно прост. Но обратного пути, как правило, нет — трассировка растра, при том что требует значительных вычислительных мощностей и времени, не всегда обеспечивает высокое качество векторного рисунка.

Преимущество векторной картинки — масштабируемость — пропадает, когда начинаем иметь дело с особо малыми разрешениями графики (например, иконки 32×32 или 16×16). Чтобы не было «грязи», картинку под такие разрешения приходится подгонять вручную.

Форматов файлов векторной графики существует намного меньше, чем растровой. Приведем примеры самых распространенных из них.

WMF (англ. Windows MetaFile - метафайл Windows) - универсальный формат для Windows-дополнений. Используется для хранения коллекции графических изображений Microsoft Clip Gallery. Основные недостатки - искажение цвета, невозможность сохранения ряда дополнительных параметров объектов.

CGM (англ. Computer Graphic Metafile - метафайл компьютерной графики) - широко использует стандартный формат векторных графических данных в сети Internet.

CDR (англ. CorelDRaw files - файлы CorelDRaw) - формат, который используется в векторном графическом редакторе Corel Draw.

AI - формат, который поддерживается векторным редактором Adobe Illustrator.

Существуют *универсальные форматы графических файлов*, которые одновременно поддерживают и векторные, и растровые изображения. Формат **PDF** (англ. Portable Document Format - портативный формат документа) разработан для работы с пакетом программ Acrobat. В этом формате могут быть сохранены изображения и векторного, и растрового формата, текст с большим количеством шрифтов, гипертекстовые ссылки и даже настройки печатающего устройства. Размеры файлов достаточно малы.

Формат **EPS** (англ. Encapsulated PostScript - инкапсулированный постскриптум) - формат, который поддерживается программами для разных операционных систем. Рекомендуется для печати и создания иллюстраций в настольных издательских системах. Этот формат позволяет сохранить векторный контур, который будет ограничивать растровое изображение.

Векторные графические редакторы позволяют пользователю создавать и редактировать векторные изображения непосредственно на экране компьютера, а также сохранять их в различных векторных форматах.

– *Основные инструменты векторных редакторов*
– Кривые Безье — позволяют создавать прямые, ломаные и гладкие кривые, проходящие через узловые точки, с определёнными касательными в этих точках;

- Заливка — позволяет закрашивать ограниченные области определённым цветом или градиентом;
- Текст создаётся с помощью соответствующего инструмента, а потом часто преобразуется в кривые, чтобы обеспечить независимость изображения от шрифтов, имеющихся (или отсутствующих) на компьютере, используемом для просмотра;
- Набор геометрических примитивов;
- Карандаш — позволяет создавать линии «от руки». При создании таких линий возникает большое количество узловых точек, от которых в дальнейшем можно избавиться с помощью «упрощения кривой».

Сравнение растровых и векторных редакторов

Векторные редакторы часто противопоставляют растровым редакторам. В действительности, их возможности часто дополняют друг друга:

- Векторные редакторы обычно более пригодны для создания разметки страниц, типографики, логотипов, sharp-edged artistic иллюстраций (например, мультипликация, clip art, сложные геометрические шаблоны), технических иллюстраций, создания диаграмм и составления блок-схем.
- Растровые редакторы больше подходят для обработки и ретуширования фотографий, создания фотореалистичных иллюстраций, коллажей, и создания рисунков от руки с помощью графического планшета.

В качестве примеров редакторов векторной графики можно привести CorelDRAW, Adobe Illustrator, Xara Xtreme, Adobe Fireworks, Inkscape.

Фрактальная графика

Третий тип двухмерной графики - фрактальная. Фрактал - это рисунок, который состоит из подобных между собой элементов. Существует большое количество графических изображений, которые являются фракталами: треугольник Серпинского, снежинка Коха, "дракон" Хартера-Хейтуея, множество Мандельброта. Построение фрактального рисунка осуществляется по какому-то алгоритму или путём автоматической генерации изображений при помощи вычислений по конкретным формулам. Изменения значений в алгоритмах или коэффициентов в формулах приводит к модификации этих изображений. Главным преимуществом фрактальной графики есть то, что в файле фрактального изображения сохраняются только алгоритмы и формулы.

Фрактальная графика отличается тем, что никакие объекты не хранятся в её памяти. Они создаются с помощью формул и уравнений. Изменяя коэффициенты уравнения можно создать совершенно новую картину. Простейшим объектом фрактальной графики является фрактальный треугольник. Согласно заданному математическому алгоритму создаётся изображение. Новые объекты строятся, наследуя свойства родительских структур.

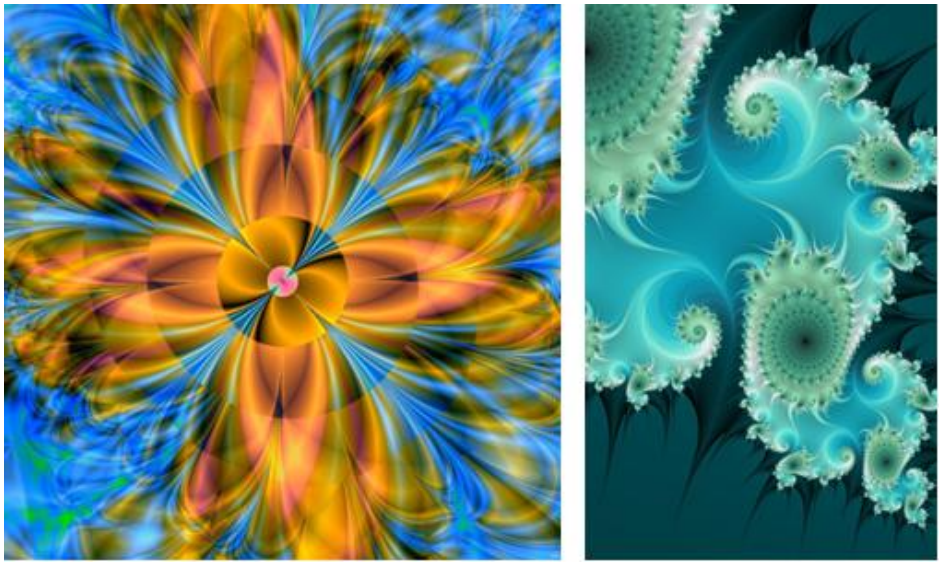


Рисунок 4.14 – Примеры фрактальной графики

Фракталы широко применяются в компьютерной графике для построения изображений природных объектов, таких как деревья, кусты, горные ландшафты, поверхности морей и так далее. Существует множество программ, служащих для генерации фрактальных изображений. Генератор фракталов — это компьютерная программа, генерирующая изображения фракталов. Большинство подобных программ позволяют выбрать алгоритм генерации фрактала, увеличить тот или иной фрагмент изображения, поменять цветовую гамму, редактировать некоторые топологические параметры и сохранять полученное изображение в одном из популярных графических форматов, таких как JPEG, TIFF или PNG, а также хранить параметры генерации конкретного фрактала, что позволяет повторное использование и модификацию таких фрактальных изображений.

Многие программы позволяют также вводить собственные формулы, и осуществлять дополнительный контроль, вроде фильтрации полученного изображения. Некоторые пакеты позволяют генерировать фрактальную анимацию.

Ряд графических редакторов общего назначения, например GIMP, включают фильтры или плагины для генерации фракталов.

Трёхмерная графика (3D Graphics, от англ. 3 Dimensions – 3 измерения) — раздел компьютерной графики, совокупность приемов и инструментов (как программных, так и аппаратных), предназначенных для изображения объёмных объектов.

Трёхмерное изображение на плоскости отличается от двухмерного тем, что включает построение геометрической проекции трёхмерной модели сцены на плоскость (например, экран компьютера) с помощью специализированных программ (однако, с созданием и внедрением 3D-дисплеев и 3D-принтеров, трёхмерная графика не обязательно включает в себя проецирование на плоскость). При этом модель может как

соответствовать объектам из реального мира (автомобили, здания, ураган, астероид), так и быть полностью абстрактной (проекция четырёхмерного фрактала).

Рисунок 4.15 – Пример объекта, смоделированного при помощи редактора трехмерной графики

Рисунок 4.16 – Пример интерьера, смоделированного при помощи редактора трехмерной графики

Трёхмерная графика активно применяется для создания изображений на плоскости экрана или листа печатной продукции в науке и промышленности, например в системах автоматизации проектных работ (САПР; для создания твердотельных элементов: зданий, деталей машин, механизмов), архитектурной визуализации (сюда относится и так называемая «виртуальная археология»), в современных системах медицинской визуализации. Самое широкое применение — во многих современных компьютерных играх.

Создание

Для получения трёхмерного изображения на плоскости требуются следующие шаги:

- моделирование — создание трёхмерной математической модели сцены и объектов в ней;
- текстурирование — назначение поверхностям моделей растровых или процедурных текстур (подразумевает также настройку свойств материалов — прозрачность, отражения, шероховатость и пр.);
- освещение — установка и настройка источников света;
- анимация (в некоторых случаях) — придание движения объектам;
- динамическая симуляция (в некоторых случаях) — автоматический расчёт взаимодействия частиц, твёрдых/мягких тел и пр. с моделируемыми силами гравитации, ветра, выталкивания и др., а также друг с другом;
- рендеринг (визуализация) — построение проекции в соответствии с выбранной физической моделью;
- вывод полученного изображения на устройство вывода — дисплей или принтер.

Программные пакеты, позволяющие создавать трёхмерную графику, то есть моделировать объекты виртуальной реальности и создавать на основе этих моделей изображения, очень разнообразны. Последние годы устойчивыми лидерами в этой области являются коммерческие продукты, такие как Autodesk 3D Studio Max и Autodesk Maya.

Форматы файлов, используемые в трехмерной графике: COLLADA (формат, разработанный для обмена между 3D приложениями), SKP (формат файлов программы SketchUp, предназначенной для моделирования относительно простых трёхмерных объектов), STL (stereolithography format, стереолитография), U3D (Universal 3D file format, универсальный формат файла трёхмерной графики, использующий сжатие данных), VRML (Virtual Reality Modeling Language, стандартный формат файлов для демонстрации трёхмерной интерактивной векторной графики).

Аппаратное обеспечение

3D-сканер — устройство, анализирующее физический объект и на основе полученных данных создающее его 3D-модель.

Полученные методом сканирования 3D-модели в дальнейшем могут быть обработаны средствами САПР и, в дальнейшем, могут использоваться для разработки технологии изготовления (CAM) и инженерных расчётов (CAE). Для вывода 3D-моделей могут использоваться такие средства, как 3D-монитор и 3D-принтер.

Рисунок 4.17 – 3D-сканер

3D-принтер — устройство, использующее метод послойного создания физического объекта на основе виртуальной 3D-модели.

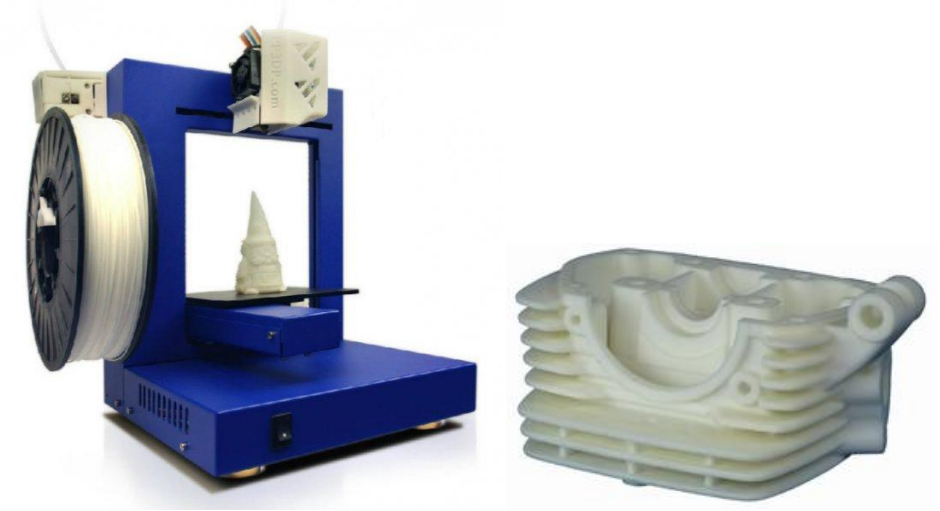


Рисунок 4.18 – 3D-принтер и пример напечатанного изделия

4.5 Введение в информационную безопасность

Как уже многократно говорилось ранее в учебном пособии, информация может иметь различную форму, включая цифровые данные, письма или памятные записки, досье, формулы, чертежи, диаграммы, модели продукции и прототипы, диссертации, судебные документы и другое. Как и всякий продукт, информация имеет потребителей, нуждающихся в ней, и потому обладает определенными потребительскими качествами, а также имеет и своих обладателей или производителей. С точки зрения потребителя, качество используемой информации позволяет получать дополнительный экономический или моральный эффект. С точки зрения обладателя — сохранение в тайне коммерчески важной информации позволяет успешно конкурировать на рынке производства и сбыта товаров и услуг. Это, естественно, требует определенных действий, направленных на защиту конфиденциальной информации. Понимая под безопасностью состояние защищенности жизненно важных интересов личности, предприятия, государства от внутренних и внешних угроз, можно выделить и компоненты

безопасности — такие, как персонал, материальные и финансовые средства и информацию.

Информация – это абстрактное содержание какого-либо высказывания, описание, указание, сообщение или известие.

Защищаемой информацией называют информацию, являющуюся предметом собственности и подлежащую защите в соответствии с требованиями правовых документов или требованиями, установленными собственником информации.

Однако защите подлежит не всякая информация, а только та, которая имеет цену. Ценной становится та информация, обладание которой позволит ее существующему и потенциальному владельцу получить какой-либо выигрыш: моральный, материальный, политический и т. д.

Ценность информации является критерием при принятии любого решения о ее защите и для выбора метода защиты. В денежном выражении затраты на защиту информации не должны превышать возможные потери.

Принято следующее разделение информации по уровню важности:

– *жизненно важная незаменимая информация*, наличие которой необходимо для функционирования организации;

– *важная информация* – информация, которая может быть заменена или восстановлена, но процесс восстановления очень труден и связан с большими затратами;

– *полезная информация* – информация, которую трудно восстановить, однако организация может эффективно функционировать и без нее;

– *несущественная информация* – информация, которая больше не нужна организации.

Категория важности, как и ценность информации, обычно изменяется со временем и зависит от степени отношения к ней различных групп потребителей и потенциальных нарушителей. Приведенное деление информации по уровню важности согласуется с принципом деления информации по уровням секретности.

Уровень секретности – это административная или законодательная мера, соответствующая мере ответственности лица за утечку или потерю конкретной секретной информации, регламентируемой специальным документом, с учетом государственных, военно-стратегических, коммерческих, служебных или частных интересов. Такой информацией может быть государственная, военная, коммерческая, служебная или личная тайна.

Характер и формы угроз

Угроза информационной безопасности – целенаправленное действие, которое повышает уязвимость накапливаемой, хранимой и обрабатываемой информации и приводит к ее случайному или предумышленному изменению или уничтожению.

Основными угрозами информационной безопасности являются:

1. Деятельность человека, непосредственно и опосредовано влияющая на информационную безопасность и являющаяся основным источником угроз.
2. Отказы и неисправности средств информатизации.
3. Стихийные бедствия и катастрофы.

Источники угроз информационной безопасности делятся на *внешние* и *внутренние*.

Внешние угрозы исходят от природных явлений (стихийных бедствий), катастроф, а также от субъектов, не входящих в состав пользователей и обслуживающего персонала системы, разработчиков системы, и не имеющих непосредственного контакта с информационными системами и ресурсами. Источники внешних угроз:

- недружественная политика сторонних лиц и организаций в области распространения информации и новых информационных технологий;
- преступные действия групп, формирований и отдельных лиц, направленные против экономических интересов организации;
- стихийные бедствия и катастрофы.

Внутренние угрозы исходят от пользователей и обслуживающего персонала системы, разработчиков системы, других субъектов, вовлеченных в информационные процессы и имеющих непосредственный контакт с информационными системами и ресурсами, как допущенных, так и не допущенных к секретным (конфиденциальным) сведениям. Источники внутренних угроз:

- противозаконная деятельность сотрудников, отделов и служб в области формирования, распространения и использования информации;
- нарушения установленных регламентов сбора, обработки и передачи информации;
- преднамеренные действия и непреднамеренные ошибки персонала информационных систем;
- отказы технических средств и сбои программного обеспечения в информационных и телекоммуникационных системах.

Реализуются угрозы по отношению к защищаемым объектам возможными способами нарушения информационной безопасности, которые могут быть разделены на информационные, программно-математические, физические, радиоэлектронные и организационно-правовые.

Информационные способы нарушения информационной безопасности:

- противозаконный сбор, распространение и использование информации;
- манипулирование информацией (дезинформация, сокрытие или искажение информации);
- незаконное копирование, уничтожение данных и программ;

- хищение информации из баз и банков данных;
- нарушение адресности и оперативности информационного обмена;
- нарушение технологии обработки данных и информационного обмена.

Программно-математические способы нарушения информационной безопасности:

- внедрение программ-вирусов;
- внедрение программных закладок на стадии проектирования или эксплуатации системы и приводящих к компрометации системы защиты информации.

Физические способы нарушения информационной безопасности:

- уничтожение, хищение и разрушение средств обработки и защиты информации, средств связи, целенаправленное внесение в них неисправностей;
- уничтожение, хищение и разрушение машинных или других оригиналов носителей информации;
- хищение ключей (ключевых документов) средств криптографической защиты информации, программных или аппаратных ключей средств защиты информации от несанкционированного доступа;
- воздействие на обслуживающий персонал и пользователей системы с целью создания благоприятных условий для реализации угроз информационной безопасности;
- диверсионные действия по отношению к объектам информационной безопасности.

Радиоэлектронные способы нарушения информационной безопасности:

- перехват информации в технических каналах ее утечки;
- перехват и дешифрование информации в сетях передачи данных и линиях связи;
- внедрение электронных устройств перехвата информации в технические средства и помещения;
- навязывание ложной информации по сетям передачи данных и линиям связи;
- радиоэлектронное подавление линий связи и систем управления с использованием одноразовых и многоразовых генераторов различных видов электромагнитной энергии (взрывомагнитные, взрывные магнитогидродинамические, пучково-плазменные и др.).

Организационно-правовые способы нарушения информационной безопасности:

- закупка несовершенных, устаревших или неперспективных средств информатизации и информационных технологий;
- невыполнение требований законодательства и задержки в разработке и принятии необходимых нормативных правовых и технических документов в области информационной безопасности.

Результатами реализации угроз информационной безопасности и осуществления посягательств (способов воздействия) на информационные ресурсы и информационные системы и процессы в общем случае являются:

- нарушение секретности (конфиденциальности) информации (разглашение, утрата, хищение, утечка и перехват и т.д.);
- нарушение целостности информации (уничтожение, искажение, подделка и т.д.);
- нарушение доступности информации и работоспособности информационных систем (блокирование данных и информационных систем, разрушение элементов информационных систем, компрометация системы защиты информации и т.д.).

Модель нарушителя информационной безопасности

В качестве вероятного нарушителя информационной безопасности рассматривается субъект, имеющий возможность реализовывать, в том числе с помощью технических средств, угрозы информационной безопасности и осуществлять посягательства (способы воздействия) на информационные ресурсы и системы.

По уровню возможности реализовать угрозы и осуществить посягательства на информационные ресурсы и системы нарушителей можно классифицировать следующим образом:

1-ый уровень – внешний нарушитель (группа внешних нарушителей), самостоятельно осуществляющий создание методов и средств реализации угроз, а также реализующий угрозы (атаки);

2-ой уровень – внутренний нарушитель, не являющийся пользователем информационных систем (группа нарушителей, среди которых есть по крайней мере один указанный выше внутренний нарушитель), самостоятельно осуществляющий создание методов и средств реализации угроз, а также реализующий угрозы (атаки);

3-ий уровень – внутренний нарушитель, являющийся пользователем информационных систем (группа нарушителей, среди которых есть по крайней мере один указанный выше внутренний нарушитель), самостоятельно осуществляющий создание методов и средств реализации угроз, а также реализующий угрозы (атаки);

4-ый уровень – группа нарушителей (среди которых есть внутренние, являющиеся пользователями информационных систем), осуществляющая создание методов и средств реализации угроз, а также реализующая их с привлечением отдельных специалистов, имеющих опыт разработки и анализа средств защиты информации, используемых в информационных системах таможенных органов;

5-ый уровень – группа нарушителей (среди которых есть внутренние, являющиеся пользователями информационных систем), осуществляющая создание методов и средств реализации атак, а также реализующая атаки с привлечением научно- исследовательских центров, специализирующихся в области разработки и анализа средств защиты информации (включая специалистов в области использования для реализации угроз (атак) недокументированных средств прикладного программного обеспечения);

Предполагается, что на этих уровнях нарушитель является специалистом высшей квалификации, знает все об информационной системе, о системе и средствах ее защиты и может при определенных обстоятельствах осуществить весь спектр посягательств на информационные ресурсы.

В своей противоправной деятельности вероятный нарушитель может использовать любое существующее в стране и за рубежом средство перехвата информации, воздействия на информацию и информационные системы, адекватные финансовые средства для подкупа персонала, шантаж и другие средства и методы для достижения стоящих перед ним целей.

Необходимо учитывать также цели посягательств вероятного нарушителя на информационные ресурсы и системы. Среди таких целей может быть хищение информации (шпионаж, в том числе экономический), намерение совершить корыстное преступление, любопытство, удовлетворение собственного тщеславия, месть, вандализм и др.

Для различных объектов обеспечения информационной безопасности модель нарушителя может быть различной и уточняется по мере необходимости.

Формы обеспечения информационной безопасности

Для предохранения информации от несанкционированного доступа выделяют следующие формы защиты:

- физические (препятствие);
- законодательные;
- управление доступом;
- криптографическое закрытие.

Физические формы защиты основаны на создании физических препятствий для злоумышленника, преграждающих ему путь к защищаемой информации (строгая система пропуска на территорию и в помещения с аппаратурой или с носителями информации). Эти способы дают защиту только от "внешних" злоумышленников и не защищают информацию от тех лиц, которые обладают правом входа в помещение.

Законодательные формы защиты составляют нормативные документы, которые регламентируют правила использования и обработки информации ограниченного доступа и устанавливают меры ответственности за нарушение этих правил.

Управление доступом представляет способ защиты информации путем регулирования доступа ко всем ресурсам системы (техническим, программным, элементам баз данных). В информационных системах регламентированы порядок работы пользователей и персонала, право доступа к отдельным файлам в базах данных и т. д. Управление доступом предусматривает следующие функции защиты:

- идентификацию пользователей, персонала и ресурсов системы (присвоение каждому объекту персонального идентификатора: имени, кода, пароля и т.п.);
- аутентификацию – опознание (установление подлинности) объекта или субъекта по предъявляемому им идентификатору;
- авторизацию – проверку полномочий (проверку соответствия дня недели, времени суток, запрашиваемых ресурсов и процедур установленному регламенту);
- разрешение и создание условий работы в пределах установленного регламента;
- регистрацию (протоколирование) обращений к защищаемым ресурсам;
- реагирование (сигнализация, отключение, задержка работ, отказ в запросе) при попытках несанкционированных действий.

Установление подлинности пользователя

Самым распространенным методом установления подлинности является метод паролей. Пароль представляет собой строку символов, которую пользователь должен ввести в систему каким-либо способом (напечатать, набрать на клавиатуре и т. п.). Если введенный пароль или его образ соответствует хранящемуся в памяти, то пользователь получает доступ ко всей информации, защищенной этим паролем. Пароль можно использовать и независимо от пользователя для защиты файлов, записей, полей данных внутри записей и т.д.

Парольная защита широко применяется в системах защиты информации и характеризуется простотой и дешевизной реализации, малыми затратами машинного времени, не требует больших объемов памяти. Однако парольная защита часто не дает достаточного эффекта по следующим причинам.

1. Чрезмерная длина пароля, не позволяющая его запомнить, стимулирует пользователя к записи пароля на подручных бумажных носителях, что сразу делает пароль уязвимым.
2. Пользователи склонны к выбору тривиальных паролей, которые можно подобрать после небольшого числа попыток перебора.

3. Процесс ввода пароля в систему поддается наблюдению даже в том случае, когда вводимые символы не отображаются на экране.
4. Таблица паролей, которая входит обычно в состав программного обеспечения операционной системы, может быть изменена, что нередко и происходит. Поэтому таблица паролей должна быть зашифрована, а ключ алгоритма шифрования должен находиться только у лица, отвечающего за безопасность информации.
5. В систему может быть внесена программа класса "троянский конь", перехватывающая вводимые пароли и записывающая их в отдельный файл, поэтому при работе с новыми программными продуктами необходима большая осторожность.

При работе с паролями рекомендуется применение следующих правил и мер предосторожности:

- не печатать пароли и не выводить их на экран;
- часто менять пароли – чем дольше используется один и тот же пароль, тем больше вероятность его раскрытия;
- каждый пользователь должен хранить свой пароль и не позволять посторонним узнать его;
- всегда зашифровывать пароли и обеспечивать их защиту недорогими и эффективными средствами;
- правильно выбирать длину пароля (чем она больше, тем более высокую степень безопасности будет обеспечивать система), так как труднее будет отгадать пароль.

Основным методом защиты информации от несанкционированного доступа является метод обеспечения разграничения функциональных полномочий и доступа к информации, направленный на предотвращение не только возможности потенциального нарушителя «читать» хранящуюся в ЭВМ информацию, но и возможности нарушителя модифицировать ее штатными и нештатными средствами.

Надежность защиты может быть обеспечена правильным подбором основных механизмов защиты, некоторые из них рассмотрим ниже.

Механизм регламентации, основанный на использовании метода защиты информации, создает такие условия автоматизированной обработки, хранения и передачи защищаемой информации, при которых возможности НСД к ней сводились бы к минимуму.

Механизм аутентификации. Различают одностороннюю и взаимную аутентификацию. В первом случае один из взаимодействующих объектов проверяет подлинность другого, тогда как во втором случае проверка является взаимной.

Криптографические методы защиты информации. Эти методы защиты широко применяются за рубежом как при обработке, так и при хранении информации, в том числе на дискетах. Для реализации мер безопасности используются различные способы шифрования (криптографии), суть которых заключается в том, что данные, отправляемые

на хранение, или сообщения, готовые для передачи, зашифровываются и тем самым преобразуются в шифрограмму или закрытый текст.

Санкционированный пользователь получает данные или сообщение, дешифрует их или раскрывает посредством обратного преобразования криптограммы, в результате чего получается исходный открытый текст.

Методу преобразования в криптографической системе соответствует использование специального алгоритма. Действие такого алгоритма запускается уникальным числом (или битовой последовательностью), обычно называемым шифрующим ключом.

В современной криптографии существуют два типа криптографических алгоритмов:

1) классические алгоритмы, основанные на использовании закрытых, секретных ключей (симметричные);

2) алгоритмы с открытым ключом, в которых используются один открытый и один закрытый ключ (асимметричные). В настоящее время находят широкое практическое применение в средствах защиты электронной информации алгоритмы с секретным ключом.

Методы криптографической защиты

Симметричное шифрование

Симметричное шифрование применяемое в классической криптографии, предполагает использование всего лишь одной секретной единицы – ключа, который позволяет отправителю зашифровать сообщение, а получателю расшифровать его. В случае шифрования данных, хранимых на магнитных или иных носителях информации, ключ позволяет зашифровать информацию при записи на носитель и расшифровать при чтении с него. Стойкость хорошей шифровальной системы определяется лишь секретностью ключа.

Все многообразие существующих криптографических методов сводят к следующим классам преобразований.

Моно- и многоалфавитные подстановки – наиболее простой вид преобразований, заключающийся в замене символов исходного текста на другие (того же алфавита) по более или менее сложному правилу. Для обеспечения высокой криптостойкости требуется использование больших ключей.

Перестановки – несложный метод криптографического преобразования, используемый, как правило, в сочетании с другими методами.

Гаммирование – метод, который заключается в наложении на открытые данные некоторой псевдослучайной последовательности, генерируемой на основе ключа.

Блочные шифры – представляют собой последовательность (с возможным повторением и чередованием) основных методов преобразования, применяемую к блоку (части) шифруемого текста. Блочные шифры на практике встречаются чаще, чем чистые преобразования того или

иного класса в силу их более высокой криптостойкости. Российский и американский стандарты шифрования основаны именно на этом классе шифров.

Метод гаммирования достаточно легко реализуем и заключается в генерации гаммы шифра с помощью генератора псевдослучайных чисел при определенном ключе и наложении полученной гаммы на открытые данные обратимым способом. Под гаммой шифра понимается псевдослучайная двоичная последовательность, вырабатываемая 'EDи по заданному алгоритму, для шифрования открытых данных и расшифровывания зашифрованных данных.

Для генерации гаммы применяют программы для ЭВМ, которые называются генераторами случайных чисел. При этом требуется, чтобы, даже зная закон формирования, но не зная ключа в виде начальных условий, никто не смог бы отличить числовой ряд от случайного.

Известны три основных требования к криптографически стойкому генератору псевдослучайной последовательности или гамме.

1. Период гаммы должен быть достаточно большим для шифрования сообщений различной длины.
2. Гамма должна быть труднопредсказуемой. Это значит, что если известны тип генератора и некоторая часть гаммы, то невозможно предсказать следующий за этой частью бит гаммы с вероятностью выше определенного значения. Тогда, если криптоаналитику станет известна какая-то часть гаммы, он все же не сможет определить биты, предшествующие ей или следующие за ней.
3. Генерирование гаммы не должно быть связано с большими техническими и организационными трудностями.

Таким образом, стойкость шифрования с помощью генератора псевдослучайных чисел зависит как от характеристик генератора, так и, причем в большей степени, от алгоритма получения гаммы.

Процесс расшифровывания данных сводится к повторной генерации гаммы шифра при известном ключе и наложению такой гаммы на зашифрованные данные. Этот метод криптографической защиты реализуется достаточно легко и обеспечивает довольно высокую скорость шифрования, однако недостаточно стоек к дешифрованию и поэтому неприменим для серьезных информационных систем.

Среди других известных алгоритмов криптографической защиты информации можно назвать алгоритмы DES, Rainbow (США); FEAL-4 и FEAL-8 (Япония); B-Crypt (Великобритания); алгоритм шифрования по ГОСТ 28147-89 (Россия) и ряд других, реализованных зарубежными и отечественными поставщиками программных и аппаратных средств защиты.

Рассмотрим некоторые из них, наиболее широко применяемых в зарубежной и отечественной практике.

Алгоритм, изложенный в стандарте DES (Data Encryption Standard), был принят в качестве федерального стандарта в 1977 г., наиболее распространен

и широко применялся для шифрования данных в США. Алгоритм DES не является закрытым и был опубликован для широкого ознакомления, что позволяет пользователям свободно применять его для своих целей.

При шифровании применяется 64-разрядный ключ. Для шифрования используются только 56 разрядов ключа, а остальные восемь разрядов являются контрольными. Он обладает большой гибкостью при реализации различных приложений обработки данных, так как каждый блок данных шифруется независимо от других. Это позволяет расшифровывать отдельные блоки зашифрованных сообщений или структуры данных, а следовательно, открывает возможность независимой передачи блоков данных или произвольного доступа к зашифрованным данным. Алгоритм может реализовываться как программным, так и аппаратным способом. Существенный недостаток этого алгоритма – малая длина ключа.

Алгоритм шифрования, определяемый российским стандартом ГОСТ 28.147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования», является единым алгоритмом криптографической защиты данных для крупных информационных систем, локальных вычислительных сетей и автономных компьютеров. Этот алгоритм может реализовываться как аппаратным, так и программным способом, удовлетворяет всем криптографическим требованиям, сложившимся в мировой практике, и, как следствие, позволяет осуществлять криптографическую защиту любой информации независимо от степени ее секретности.

В алгоритме ГОСТ 28.147 – 89 в отличие от алгоритма DES используется 256-разрядный ключ, представляемый в виде восьми 32-разрядных чисел. Расшифровываются данные с помощью того же ключа, посредством которого они были зашифрованы. Алгоритм ГОСТ 28.147 – 89 полностью удовлетворяет всем требованиям криптографии и обладает теми же достоинствами, что и другие алгоритмы (например, DES), но лишен их недостатков. Он позволяет обнаруживать как случайные, так и умышленные модификации зашифрованной информации. Крупный недостаток этого алгоритма – большая сложность его программной реализации и низкая скорость работы.

Из алгоритмов шифрования, разработанных в последнее время, большой интерес представляет алгоритм RC6 фирмы RSA Data Security. Этот алгоритм обладает следующими свойствами:

- адаптивностью для аппаратных средств и программного обеспечения, что означает использование в нем только примитивных вычислительных операций, обычно присутствующих на типичных микропроцессорах;
- быстротой, т.е. в базисных вычислительных операциях операторы работают на полных словах данных;
- адаптивностью на процессоры различных длин слова. Число бит в слове – параметр алгоритма;

- наличием параметра, отвечающего за «степень перемешивания», т.е. число раундов (итераций до 255). Пользователь может явно выбирать между более высоким быстродействием и более высоким перемешиванием;
- низким требованием к памяти, что позволяет реализовывать алгоритм на устройствах с ограниченной памятью;
- использованием циклических сдвигов, зависимых от данных, с переменным числом;
- простотой и легкостью выполнения.

Методы асимметричного шифрования

Методы асимметричного шифрования предполагают шифрование при наличии двух ключей – секретного и публичного (открытого). Особенность таких методов заключается в одностороннем характере применения ключей.

Например, зная ключ шифрования можно зашифровать сообщение, но с помощью этого же ключа расшифровать его обратно невозможно.

Математическая теория асимметричного шифрования основана на применении однонаправленных функций, по результатам вычисления которых определить аргументы практически невозможно. Например, невозможно определить каким способом было получено число 16 – перемножением 1·16, 2·8, 4·4, 4·2·2 и т.п.

Электронная цифровая подпись: алгоритмы, открытый и секретный ключи, сертификаты

Закон "Об электронной цифровой подписи" определяет, что "электронная цифровая подпись – реквизит электронного документа, предназначенный для защиты данного электронного документа от подделки, с использованием закрытого ключа электронной цифровой подписи и позволяющий идентифицировать владельца сертификата ключа подписи, а также установить отсутствии искажения в электронном документе". Из этого определения видно, что электронная цифровая подпись (ЭЦП) формируется при помощи специальных математических алгоритмов на основе собственно документа и некого "закрытого ключа", позволяющего однозначно идентифицировать отправителя сообщения. Рассмотрим подробней механизм функционирования систем ЭЦП.

Открытый и закрытый ключи в электронной цифровой подписи

Электронная цифровая подпись функционирует на основе криптоалгоритмов с асимметричными (открытыми) ключами и инфраструктуры открытых ключей. В криптосистемах на основе асимметричных ключей для шифрования и дешифрования используется пара ключей – секретный и публичный ключи, уникальные для каждого пользователя, и цифровой сертификат.

Основные термины, применяемые при работе с ЭЦП: закрытый ключ – это некоторая информация, обычно длиной 256 бит, хранится в недоступном другим лицам месте на смарт-карте, touch memory. Работает закрытый ключ только в паре с открытым ключом.

Открытый ключ – используется для проверки ЭЦП получаемых документов-файлов технически это некоторая информация длиной 1024 бита. Открытый ключ работает только в паре с закрытым ключом. На открытый ключ выдается сертификат, который автоматически передается вместе с письмом, подписанным ЭЦП. Необходимо обеспечить наличие своего открытого ключа у всех, с кем предполагается обмениваться подписанными документами. Можно также удостовериться о личности, подписавшей электронной подписью документ, который получен, просмотрев его сертификат. Дубликат открытого ключа направляется в Удостоверяющий Центр (УЦ), где создана библиотека открытых ключей ЭЦП. В библиотеке УЦ обеспечивается регистрация и надежное хранение открытых ключей во избежание попыток подделки или внесения искажений.

Цифровой сертификат представляет собой расширение открытого ключа, включающего не только сам ключ, но и дополнительную информацию, описывающую принадлежность ключа, время использования, доступные криптосистемы, название УЦ и т.д.

Основная функция УЦ – распространение публичных и секретных ключей пользователей, а также верификация сертификатов. УЦ могут объединяться в цепочки. Вышестоящий (корневой) УЦ может выдать сертификат и права на выдачу ключей нижестоящему центру. Тот, в свою очередь, может выдать права еще другому нижестоящему центру и так далее, причем сертификат, выданный одним из центров, может быть верифицирован любым из серверов в цепочке. Таким образом, существует возможность установить центр распространения секретных ключей в непосредственной близости от пользователя, что решает проблему дискредитации ключа при передаче по сетям связи.

На удостоверяющих центрах лежит огромная ответственность, поскольку именно они отвечают за надежность функционирования всей инфраструктуры открытых ключей.

Методики асимметричного шифрования

За последние 20 лет получили широкое распространение криптосистемы на базе асимметричного шифрования, позволяющие не только организовать конфиденциальную передачу информации без предварительного обмена секретным ключом, но и значительно расширяющие функции криптографии, включая технологию ЭЦП.

Наибольшее распространение в мире получила криптосистема RSA. Она была предложен тремя исследователями-математиками Рональдом Райвестом (R.Rivest) , Ади Шамиром (A.Shamir) и Леонардом Эйдлманом (L.Adleman) в 1977-78 годах. Другие криптосистемы более специализированы и поддерживают не все возможности. Широко

применяются криптосистемы, в основе которых лежат алгоритмы, не являющиеся алгоритмами собственно шифрования, но реализующие только технологию ЭЦП. К их числу относятся: российские алгоритмы электронной цифровой подписи ГОСТ Р 34.10-94 и ГОСТ Р 34.10-2001; алгоритм электронной цифровой подписи DSA, входящий в принятый в США стандарт цифровой подписи Digital Signature Standard. Известна также криптосистема на базе алгоритма Диффи-Хеллмана согласования ключа, применяемого при конфиденциальной передаче информации.

В случае с ЭЦП процесс обмена сообщением выглядит следующим образом:

- отправитель получает у удостоверяющего центра секретный ключ;
- используя этот ключ, формирует электронную цифровую подпись и отправляет письмо;
- получатель при помощи публичного (общедоступного) ключа и цифрового сертификата, полученного у удостоверяющего центра, устанавливает авторство документа и отсутствие искажений.

Рассмотрим принципиальную схему выработки и проверки ЭЦП с применением алгоритмов асимметричного шифрования.

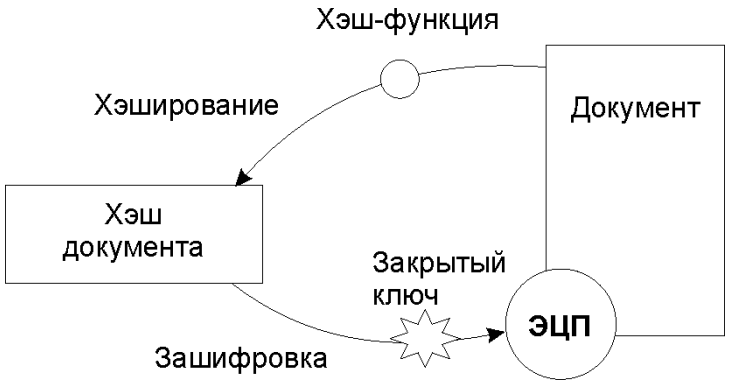


Рисунок 4.19 – Схема выработки ЭЦП при асимметричном шифровании

Для выработки ЭЦП подписываемый документ подвергается хэшированию (т.е. сжатию некоторым стандартным алгоритмом), а полученный хэш (иногда его называют дайджестом) зашифровывается закрытым ключом (рисунок 4.19). Хэширование применяется для сокращения объема шифруемой информации и повышения тем самым производительности. Хэш-функция, не будучи взаимно однозначным отображением, подбирается таким образом, чтобы было практически невозможно изменить документ, сохранив результат хэширования. По хэшу невозможно восстановить исходный документ, но это и не нужно, поскольку проверка ЭЦП заключается в сравнении расшифрованной открытым ключом ЭЦП с хэшем документа (рисунок 4.20). Совпадение с высокой степенью достоверности гарантирует, во-первых, неизменность документа (защиту от подделки), и, во-вторых, что его подписал владелец закрытого ключа.

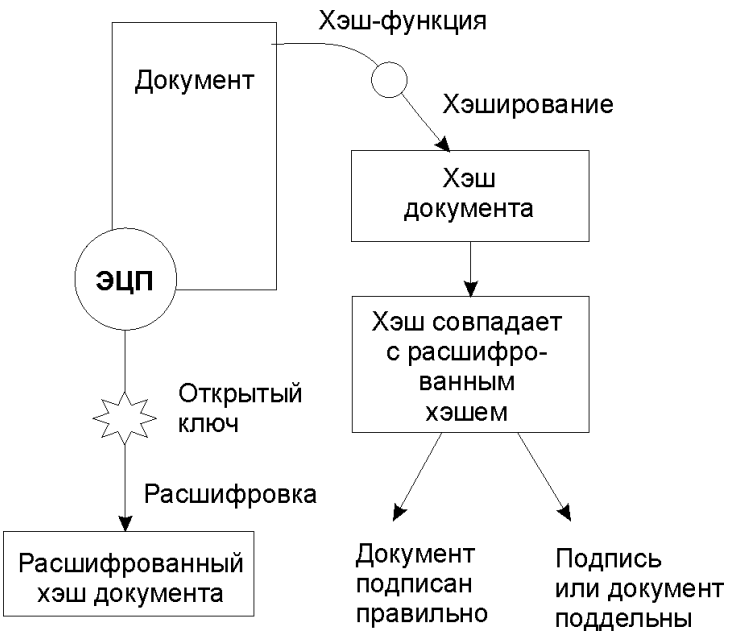


Рисунок 4.20 – Схема проверки ЭЦП при асимметричном шифровании

В специализированных криптосистемах, поддерживающих только технологию ЭЦП, функции собственно шифрования отсутствуют. Для формирования ЭЦП применяется криптоалгоритм, получающий на входе хэш документа, закрытый ключ и вырабатывающий ЭЦП. Для проверки ЭЦП применяется другой криптоалгоритм, имеющий на входе хэш документа проверяемую ЭЦП и открытый ключ. Алгоритм проверки выдает положительный или отрицательный результат в зависимости от правильности ЭЦП.

Аутентификация субъекта сводится к доказательству того, что он владеет закрытым ключом, соответствующим опубликованному открытому. В криптосистемах, поддерживающих технологию ЭЦП, доказательство владения заключается в том, что субъект подписывает своим закрытым ключом присланный ему запрос и посылает его обратно. Если при проверке оказалось, что запрос подписан правильно, то субъект действительно обладает соответствующим закрытым ключом. Необходимо принять меры, чтобы злоумышленник, перехвативший подписанный запрос, не мог впоследствии использовать его, выдавая себя за правомерного владельца закрытого ключа. Для борьбы с этим достаточно, чтобы запрос был неповторяющимся.

Асимметричные алгоритмы шифрования позволяют обеспечить конфиденциальность при передаче сообщения от одного субъекта другому. Для этого отправителю достаточно зашифровывать сообщение открытым ключом получателя. Поскольку расшифровать сообщение можно, только зная соответствующий закрытый ключ, это гарантирует, что прочитать его не сможет никто, кроме получателя.

На практике все сообщение никогда не шифруют открытым ключом. Дело в том, что производительность асимметричного шифрования существенно ниже симметричного, поэтому обычно в начале интерактивного

сеанса связи одна из сторон генерирует симметричный секретный ключ (ключ сеанса), шифрует его открытым ключом другой стороны и передает только этот зашифрованный ключ. Другая сторона принимает и расшифровывает его (очевидно, при этом сохраняется конфиденциальность), а все дальнейшие сообщения уже могут быть зашифрованы согласованным ключом сеанса. По окончании сеанса этот ключ уничтожается.

Если нужно послать сообщение вне интерактивного сеанса, то достаточно приложить к зашифрованному сообщению секретный ключ, зашифрованный открытым ключом получателя.

Специализированный алгоритм Диффи-Хелмана согласования ключа позволяет каждой стороне контакта, зная свой закрытый ключ и открытый ключ партнера, получить "общий секрет", используемый для создания единого секретного ключа, предназначенного для заранее согласованного алгоритма симметричного шифрования. Практически невозможно, зная только открытые ключи, воспроизвести "общий секрет", что гарантирует защиту от злоумышленника. При интерактивном контакте стороны сначала обмениваются открытыми ключами, а потом получают единый ключ сеанса. При посылке зашифрованного сообщения вне интерактивного сеанса отправителю должен быть известен открытый ключ получателя; к сообщению же прилагается открытый ключ отправителя, что позволяет получателю воссоздать секретный ключ. В криптосистеме на базе этого алгоритма процедура доказательства владения закрытым ключом подобна процедуре подписывания ЭЦП за тем исключением, что при подписании используется не сам закрытый ключ, а "общий секрет", зависящий еще и от открытого ключа проверяющей стороны.

Контрольные вопросы

1. Перечислите известные вам информационные категории.
2. Объясните, каким образом осуществляется автоматизация на основе алгоритмизации? Что такое реинжиниринг?
3. Перечислите отличия технологий groupware и workflow. Что такое OCR?
4. Опишите реляционную модель хранения данных, пересилите основные принципы управления данными.
5. Дайте краткую характеристику системам компьютерной графики, перечислите основные особенности и отличия различных графических систем и форматов.
6. Объясните принципы защиты информации. Что такое "модель нарушителя" ?
7. Каким образом осуществляется криптографическая защита информации?

5. ПРОГРАММНО-АППАРАТНЫЕ СРЕДСТВА РЕАЛИЗАЦИИ ИНФОРМАЦИОННЫХ ПРОЦЕССОВ

5.1 Операционные системы

Операционная система (ОС) в наибольшей степени определяет облик всей вычислительной системы в целом. Несмотря на это, пользователи, активно использующие вычислительную технику, зачастую испытывают затруднения при попытке дать определение операционной системе. Частично это связано с тем, что ОС выполняет две по существу мало связанные функции: обеспечение пользователю-программисту удобств посредством предоставления для него расширенной машины и повышение эффективности использования компьютера путем рационального управления его ресурсами.

Использование большинства компьютеров на уровне машинного языка затруднительно, особенно это касается ввода-вывода. Например, для организации чтения блока данных с гибкого диска программист может использовать 16 различных команд, каждая из которых требует 13 параметров, таких как номер блока на диске, номер сектора на дорожке и т.п. Когда выполнение операции с диском завершается, контроллер возвращает 23 значения, отражающих наличие и типы ошибок, которые, очевидно, надо анализировать. Даже если не входить в курс реальных проблем программирования ввода-вывода, ясно, что среди программистов нашлось бы немного желающих непосредственно заниматься программированием этих операций. При работе с диском программисту-пользователю достаточно представлять его в виде некоторого набора файлов, каждый из которых имеет имя. Работа с файлом заключается в его открытии, выполнении чтения или записи, а затем в закрытии файла. Вопросы подобные таким, как следует ли при записи использовать усовершенствованную частотную модуляцию или в каком состоянии сейчас находится двигатель механизма перемещения считывающих головок, не должны волновать пользователя. Программа, которая скрывает от программиста все реалии аппаратуры и предоставляет возможность простого, удобного просмотра указанных файлов, чтения или записи - это, конечно, операционная система. Точно так же, как ОС ограждает программистов от аппаратуры дискового накопителя и предоставляет ему простой файловый интерфейс, операционная система берет на себя все малоприятные дела, связанные с обработкой прерываний, управлением таймерами и оперативной памятью, а также другие низкоуровневые проблемы. В любом случае та абстрактная, воображаемая машина, с которой благодаря операционной системе теперь может иметь дело пользователь, гораздо проще и удобнее в обращении, чем реальная аппаратура, лежащая в основе этой абстрактной машины.

С этой точки зрения функцией ОС является предоставление пользователю некоторой расширенной или виртуальной машины, которую легче программировать и с которой легче работать, чем непосредственно с аппаратурой, составляющей реальную машину.

Идея о том, что ОС, прежде всего система, обеспечивающая удобный интерфейс пользователям, соответствует рассмотрению сверху вниз. Другой взгляд, снизу вверх, дает представление об ОС как о некотором механизме, управляющем всеми частями сложной системы. Современные вычислительные системы состоят из процессоров, памяти, таймеров, дисков, накопителей на магнитных лентах, сетевых коммуникационной аппаратуры, принтеров и других устройств. В соответствии со вторым подходом функцией ОС является распределение процессоров, памяти, устройств и данных между процессами, конкурирующими за эти ресурсы.

ОС должна управлять всеми ресурсами вычислительной машины таким образом, чтобы обеспечить максимальную эффективность ее функционирования. Критерием эффективности может быть, например, пропускная способность или реактивность системы. Управление ресурсами включает решение двух общих, не зависящих от типа ресурса, задач:

- планирование ресурса - то есть определение, кому, когда, а для делимых ресурсов - и в каком количестве, необходимо выделить данный ресурс;
- отслеживание состояния ресурса - то есть поддержание оперативной информации о том, занят или не занят ресурс, а для делимых ресурсов - какое количество ресурса уже распределено, а какое свободно.

Для решения этих общих задач управления ресурсами разные ОС используют различные алгоритмы, что, в конечном счете, и определяет их облик в целом, включая характеристики производительности, область применения и даже пользовательский интерфейс. Так, например, алгоритм управления процессором в значительной степени определяет, является ли ОС системой разделения времени, системой пакетной обработки или системой реального времени.

Классификация ОС

Операционные системы могут различаться особенностями реализации внутренних алгоритмов управления основными ресурсами компьютера (процессорами, памятью, устройствами),

особенностями использованных методов проектирования, типами аппаратных платформ, областями использования и многими другими свойствами.

Ниже приведена классификация ОС по нескольким наиболее основным признакам.

Особенности алгоритмов управления ресурсами

От эффективности алгоритмов управления локальными ресурсами компьютера во многом зависит эффективность всей сетевой ОС в целом. Поэтому, характеризуя сетевую ОС, часто приводят важнейшие особенности реализации функций ОС по управлению процессорами, памятью, внешними устройствами автономного компьютера. Так, например, в зависимости от

особенностей использованного алгоритма управления процессором операционные системы делят на многозадачные и однозадачные, многопользовательские и однопользовательские, на системы, поддерживающие многонитевую обработку и не поддерживающие ее, на многопроцессорные и однопроцессорные системы.

Поддержка многозадачности

По числу одновременно выполняемых задач операционные системы могут быть разделены на два класса:

- однозадачные (например, MS-DOS, MSX);
- многозадачные (ОС UNIX, Linux, Windows, Novell Netware, iOS от Apple). Однозадачные ОС в основном выполняют функцию предоставления пользователю виртуальной машины, делая более простым и удобным процесс взаимодействия пользователя с компьютером. Однозадачные ОС включают средства управления периферийными устройствами, средства управления файлами, средства общения с пользователем.

Многозадачные ОС, кроме вышеперечисленных функций, управляют разделением совместно используемых ресурсов, таких как процессор, оперативная память, файлы и внешние устройства.

Поддержка многопользовательского режима

По числу одновременно работающих пользователей ОС делятся на:

- однопользовательские (MS-DOS, Windows 3.x, ранние версии OS/2, фактически Windows 9x/Me);
- многопользовательские (UNIX, Linux, Windows NT/2000/XP/Vista/7/8, iOS).

Главным отличием многопользовательских систем от однопользовательских является наличие средств защиты информации каждого пользователя от несанкционированного доступа других пользователей. Следует заметить, что не всякая многозадачная система является многопользовательской, и не всякая однопользовательская ОС является однозадачной.

Вытесняющая, кооперативная и невытесняющая многозадачность

Важнейшим разделяемым ресурсом является процессорное время. Способ распределения процессорного времени между несколькими одновременно существующими в системе процессами (или нитями) во многом определяет специфику ОС. Среди множества существующих вариантов реализации многозадачности можно выделить три группы алгоритмов:

- невытесняющая многозадачность (программа DESQView для DOS);
- кооперативная многозадачность (NetWare, Windows 3.x, 16-битные приложения в Windows 9x/Me);
- вытесняющая многозадачность (Windows NT/2000/XP/Vista/7/8,

UNIX, iOS). Основным различием между вытесняющим и кооперативным вариантами многозадачности является степень централизации механизма планирования процессов. В первом случае механизм планирования процессов целиком сосредоточен в операционной системе, а во втором - распределен между системой и прикладными программами. При кооперативной многозадачности активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление операционной системе для того, чтобы та выбрала из очереди другой готовый к выполнению процесс. При вытесняющей многозадачности решение о переключении процессора с одного процесса на другой принимается операционной системой, а не самим активным процессом.

Поддержка многопоточности и многонитевости

Важным свойством операционных систем является возможность распараллеливания вычислений.

Многопоточность — свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

Такие потоки называют также потоками выполнения.

Сутью многопоточности является квазимногозадачность на уровне одного исполняемого процесса, то есть все потоки выполняются в адресном пространстве процесса. Кроме этого, все потоки процесса имеют не только общее адресное пространство, но и общие дескрипторы файлов. Выполняющийся процесс имеет как минимум один (главный) поток.

Многопоточность (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью, несмотря на то, что операционные системы, реализующие многозадачность, как правило реализуют и многопоточность.

К достоинствам многопоточности в программировании можно отнести следующее:

- Упрощение программы в некоторых случаях за счет использования общего адресного пространства.
- Меньшие относительно процесса временные затраты на создание потока.
- Повышение производительности процесса за счет распараллеливания процессорных вычислений и операций ввода/вывода.

Говоря о процессах, мы отмечали, что операционная система поддерживает их обособленность: у каждого процесса имеется свое виртуальное адресное пространство, каждому процессу назначаются свои ресурсы – файлы, окна, семафоры и т.д. Такая обособленность нужна для

того, чтобы защитить один процесс от другого, поскольку они, совместно используя все ресурсы машины, конкурируют с друг другом. В общем случае процессы принадлежат разным пользователям, разделяющим один компьютер, и ОС берет на себя роль арбитра в спорах процессов за ресурсы.

При мультипрограммировании повышается пропускная способность системы, но отдельный процесс никогда не может быть выполнен быстрее, чем если бы он выполнялся в однопрограммном режиме (всякое разделение ресурсов замедляет работу одного из участников за счет дополнительных затрат времени на ожидание освобождения ресурса). Однако задача, решаемая в рамках одного процесса, может обладать внутренним параллелизмом, который в принципе позволяет ускорить ее решение. Например, в ходе выполнения задачи происходит обращение к внешнему устройству, и на время этой операции можно не блокировать полностью выполнение процесса, а продолжить вычисления по другой «ветви» процесса.

Для этих целей современные ОС предлагают использовать сравнительно новый механизм – **многонитевость** *(*multithreading*)*. При этом вводится новое понятие «нить» (thread), а понятие «процесс» в значительной степени меняет смысл.

Мультипрограммирование теперь реализуется на уровне нитей, и задача, оформленная в виде нескольких нитей в рамках одного процесса, может быть выполнена быстрее за счет псевдопараллельного (или параллельного в мультипроцессорной системе) выполнения ее отдельных частей. Например, если электронная таблица была разработана с учетом возможностей многонитевой обработки, то пользователь может запросить пересчет своего рабочего листа и одновременно продолжать заполнять таблицу. Особенно эффективно можно использовать многонитевость для выполнения распределенных приложений, например, многонитевый сервер может параллельно выполнять запросы сразу нескольких клиентов.

Нити, относящиеся к одному процессу, не настолько изолированы друг от друга, как процессы в традиционной многозадачной системе, между ними легко организовать тесное взаимодействие. Действительно, в отличие от процессов, которые принадлежат разным, вообще говоря, конкурирующим приложениям, все нити одного процесса всегда принадлежат одному приложению, поэтому программист, пишущий это приложение, может заранее продумать работу множества нитей процесса таким образом, чтобы они могли взаимодействовать, а не бороться за ресурсы.

В традиционных ОС понятие «нить» тождественно понятию «процесс». В действительности часто бывает желательно иметь несколько нитей, разделяющих единое адресное пространство, но выполняющихся квазипараллельно, благодаря чему нити становятся подобными процессам (за исключением разделяемого адресного пространства).

Нити иногда называют облегченными процессами или мини-процессами. Действительно, нити во многих отношениях подобны процессам. Каждая нить выполняется строго последовательно и имеет свой

собственный программный счетчик и стек. Нити, как и процессы, могут, например, порождать нити-потомки, могут переходить из состояния в состояние. Подобно традиционным процессам (то есть процессам, состоящим из одной нити), нити могут находится в одном из следующих состояний: ВЫПОЛНЕНИЕ, ОЖИДАНИЕ и ГОТОВНОСТЬ. Пока одна нить заблокирована, другая нить того же процесса может выполняться. Нити разделяют процессор так, как это делают процессы, в соответствии с различными вариантами планирования.

Однако различные нити в рамках одного процесса не настолько независимы, как отдельные процессы. Все такие нити имеют одно и то же адресное пространство. Это означает, что они разделяют одни и те же глобальные переменные. Поскольку каждая нить может иметь доступ к каждому виртуальному адресу, одна нить может использовать стек другой нити. Между нитями нет полной защиты, потому что, во-первых, это невозможно, а во-вторых, не нужно.

Все нити одного процесса всегда решают общую задачу одного пользователя, и аппарат нитей используется здесь для более быстрого решения задачи путем ее распараллеливания. При этом программисту очень важно получить в свое распоряжения удобные средства организации взаимодействия частей одной задачи. Кроме разделения адресного пространства, все нити разделяют также набор открытых файлов, таймеров, сигналов и т.п.

Итак, нити имеют собственные: программный счетчик; стек; регистры; нити-потомки; состояние.

Нити разделяют: адресное пространство; глобальные переменные; открытые файлы; таймеры; семафоры; статистическую информацию.

Многонитевая обработка повышает эффективность работы системы по сравнению с многозадачной обработкой. Например, в многозадачной среде Windows можно одновременно работать с электронной таблицей и текстовым редактором. Однако, если пользователь запрашивает пересчет своего рабочего листа, электронная таблица блокируется до тех пор, пока эта операция не завершится, что может потребовать значительного времени. В многонитевой среде в случае, если электронная таблица была разработана с учетом возможностей многонитевой обработки, предоставляемых программисту, этой проблемы не возникает, и пользователь всегда имеет доступ к электронной таблице.

Многопроцессорная обработка

Другим важным свойством ОС является отсутствие или наличие в ней средств поддержки многопроцессорной обработки - мультипроцессирование. Мультипроцессирование приводит к усложнению всех алгоритмов управления ресурсами.

В наши дни становится общепринятым введение в ОС функций поддержки многопроцессорной обработки данных. Такие функции имеются в операционных системах Solaris фирмы Sun, Open Server компании Santa Crus

Operations, Windows NT/2000/XP/Vista/7/8 фирмы Microsoft, iOS от Apple и NetWare фирмы Novell.

Многопроцессорные ОС могут классифицироваться по способу организации вычислительного процесса в системе с многопроцессорной архитектурой: асимметричные ОС и симметричные ОС. Асимметричная ОС целиком выполняется только на одном из процессоров системы, распределяя прикладные задачи по остальным процессорам. Симметричная ОС полностью децентрализована и использует весь пул процессоров, разделяя их между системными и прикладными задачами.

Выше были рассмотрены характеристики ОС, связанные с управлением только одним типом ресурсов - процессором. Важное влияние на облик операционной системы в целом, на возможности ее использования в той или иной области оказывают особенности и других подсистем управления локальными ресурсами - подсистем управления памятью, файлами, устройствами ввода-вывода.

Специфика ОС проявляется и в том, каким образом она реализует сетевые функции: распознавание и перенаправление в сеть запросов к удаленным ресурсам, передача сообщений по сети, выполнение удаленных запросов. При реализации сетевых функций возникает комплекс задач, связанных с распределенным характером хранения и обработки данных в сети: ведение справочной информации обо всех доступных в сети ресурсах и серверах, адресация взаимодействующих процессов, обеспечение прозрачности доступа, тиражирование данных, согласование копий, поддержка безопасности данных.

Особенности аппаратных платформ

На свойства операционной системы непосредственное влияние оказывают аппаратные средства, на которые она ориентирована. По типу аппаратуры различают операционные системы персональных компьютеров, мини-компьютеров, мейнфреймов, кластеров и сетей ЭВМ. Среди перечисленных типов компьютеров могут встречаться как однопроцессорные варианты, так и многопроцессорные. В любом случае специфика аппаратных средств, как правило, отражается на специфике операционных систем.

Очевидно, что ОС большой машины является более сложной и функциональной, чем ОС персонального компьютера. Так, в ОС больших машин функции по планированию потока выполняемых задач, очевидно, реализуются путем использования сложных приоритетных дисциплин и требуют большей вычислительной мощности, чем в ОС персональных компьютеров. Аналогично обстоит дело и с другими функциями.

Сетевая ОС имеет в своем составе средства передачи сообщений между компьютерами по линиям связи, которые совершенно не нужны в автономной ОС. На основе этих сообщений сетевая ОС поддерживает разделение ресурсов компьютера между удаленными пользователями, подключенными к сети. Для поддержания функций передачи сообщений сетевые ОС содержат специальные программные компоненты, реализующие

популярные коммуникационные протоколы, такие как TCP/IP, IPX, Ethernet и другие.

Многопроцессорные системы требуют от операционной системы особой организации, с помощью которой сама операционная система, а также поддерживаемые ею приложения могли бы выполняться параллельно отдельными процессорами системы. Параллельная работа отдельных частей ОС создает дополнительные проблемы для разработчиков ОС, так как в этом случае гораздо сложнее обеспечить согласованный доступ отдельных процессов к общим системным таблицам, исключить эффект гонок и прочие нежелательные последствия асинхронного выполнения работ.

Другие требования предъявляются к операционным системам кластеров. Кластер - слабо связанная совокупность нескольких вычислительных систем, работающих совместно для выполнения общих приложений и представляющихся пользователю единой системой. Наряду со специальной аппаратурой для функционирования кластерных систем необходима и программная поддержка со стороны операционной системы, которая сводится в основном к синхронизации доступа к разделяемым ресурсам, обнаружению отказов и динамической реконфигурации системы. Одной из первых разработок в области кластерных технологий были решения компании Digital Equipment на базе компьютеров VAX.

Наряду с ОС, ориентированными на совершенно определенный тип аппаратной платформы, существуют операционные системы, специально разработанные таким образом, чтобы они могли быть легко перенесены с компьютера одного типа на компьютер другого типа, так называемые мобильные ОС. Наиболее ярким примером такой ОС является популярная система UNIX. В этих системах аппаратно-зависимые места тщательно локализованы, так что при переносе системы на новую платформу переписываются только они. Средством, облегчающим перенос остальной части ОС, является написание ее на машинно-независимом языке, например, на С, который и был разработан для программирования операционных систем.

Особенности областей использования

Многозадачные ОС подразделяются на три типа в соответствии с использованными при их разработке критериями эффективности:

- системы пакетной обработки;
- системы разделения времени;
- системы реального времени.

Системы пакетной обработки предназначались для решения задач, в основном, вычислительного характера, не требующих быстрого получения результатов. Главной целью и критерием эффективности систем пакетной обработки является максимальная пропускная способность, то есть решение максимального числа задач в единицу времени. Для достижения этой цели в системах пакетной обработки используется следующая схема функционирования: в начале работы формируется пакет заданий, каждое

задание содержит требование к системным ресурсам; из этого пакета заданий формируется мультипрограммная смесь, то есть множество одновременно выполняемых задач. Для одновременного выполнения выбираются задачи, предъявляющие отличающиеся требования к ресурсам, так, чтобы обеспечивалась сбалансированная загрузка всех устройств вычислительной машины, например, в мультипрограммной смеси желательно одновременное присутствие вычислительных задач и задач с интенсивным вводом-выводом. Таким образом, выбор нового задания из пакета заданий зависит от внутренней ситуации, складывающейся в системе, то есть выбирается «выгодное» задание. Следовательно, в таких ОС невозможно гарантировать выполнение того или иного задания в течение определенного периода времени. В системах пакетной обработки переключение процессора с выполнения одной задачи на выполнение другой происходит только в случае, если активная задача сама отказывается от процессора, например, из-за необходимости выполнить операцию ввода-вывода. Поэтому одна задача может надолго занять процессор, что делает невозможным выполнение интерактивных задач. Таким образом, взаимодействие пользователя с вычислительной машиной, на которой установлена система пакетной обработки, сводится к тому, что он приносит задание, отдает его диспетчеру-оператору, а в конце дня после выполнения всего пакета заданий получает результат. Очевидно, что такой порядок снижает эффективность работы пользователя.

Системы разделения времени призваны исправить основной недостаток систем пакетной обработки - изоляцию пользователя-программиста от процесса выполнения его задач. Каждому пользователю системы разделения времени предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системах разделения времени каждой задаче выделяется только квант процессорного времени, ни одна задача не занимает процессор надолго, и время ответа оказывается приемлемым. Если квант выбран достаточно небольшим, то у всех пользователей, одновременно работающих на одной и той же машине, складывается впечатление, что каждый из них единолично использует машину. Ясно, что системы разделения времени обладают меньшей пропускной способностью, чем системы пакетной обработки, так как на выполнение принимается каждая запущенная пользователем задача, а не та, которая «выгодна» системе, и, кроме того, имеются накладные расходы вычислительной мощности на более частое переключение процессора с задачи на задачу. Критерием эффективности систем разделения времени является не максимальная пропускная способность, а удобство и эффективность работы пользователя.

Системы реального времени применяются для управления различными техническими объектами, такими, например, как станок, спутник, научная экспериментальная установка, или технологическими процессами, такими, как гальваническая линия, доменный процесс и т.п. Во всех этих случаях существует предельно допустимое время, в течение которого должна быть

выполнена та или иная программа, управляющая объектом, в противном случае может произойти авария: спутник выйдет из зоны видимости, экспериментальные данные, поступающие с датчиков, будут потеряны, толщина гальванического покрытия не будет соответствовать норме. Таким образом, критерием эффективности для систем реального времени является их способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата (управляющего воздействия). Это время называется временем реакции системы, а соответствующее свойство системы - реактивностью. Для этих систем мультипрограммная смесь представляет собой фиксированный набор заранее разработанных программ, а выбор программы на выполнение осуществляется исходя из текущего состояния объекта или в соответствии с расписанием плановых работ.

Некоторые операционные системы могут совмещать в себе свойства систем разных типов, например, часть задач может выполняться в режиме пакетной обработки, а часть - в режиме реального времени или в режиме разделения времени. В таких случаях режим пакетной обработки часто называют фоновым режимом.

Особенности методов построения

При описании операционной системы часто указываются особенности ее структурной организации и основные концепции, положенные в ее основу.

К таким базовым концепциям относятся:

- способы построения ядра системы - монолитное ядро или микроядерный подход. Большинство ОС использует монолитное ядро, которое компонуется как одна программа, работающая в привилегированном режиме и использующая быстрые переходы с одной процедуры на другую, не требующие переключения из привилегированного режима в пользовательский и наоборот. Альтернативой является построение ОС на базе микроядра, работающего также в привилегированном режиме и выполняющего только минимум функций по управлению аппаратурой, в то время как функции ОС более высокого уровня выполняют специализированные компоненты ОС-серверы, работающие в пользовательском режиме. При таком построении ОС работает более медленно, так как часто выполняются переходы между привилегированным режимом и пользовательским, зато система получается более гибкой - ее функции можно наращивать, модифицировать или сужать, добавляя, модифицируя или исключая серверы пользовательского режима. Кроме того, серверы хорошо защищены друг от друга, как и любые пользовательские процессы;
- построение ОС на базе объектно-ориентированного подхода дает возможность использовать все его достоинства, хорошо зарекомендовавшие себя на уровне приложений, внутри операционной системы, а именно: аккумуляцию удачных решений в форме стандартных объектов, возможность создания новых объектов на базе имеющихся с помощью механизма

наследования, хорошую защиту данных за счет их инкапсуляции во внутренние структуры объекта, что делает данные недоступными для несанкционированного использования извне, структуризованность системы, состоящей из набора хорошо определенных объектов;

- наличие нескольких прикладных сред дает возможность в рамках одной ОС одновременно выполнять приложения, разработанные для нескольких ОС. Многие современные операционные системы поддерживают одновременно прикладные среды Windows, UNIX, Linux или хотя бы некоторого подмножества из этого популярного набора. Концепция множественных прикладных сред наиболее просто реализуется в ОС на базе микроядра, над которым работают различные серверы, часть которых реализуют прикладную среду той или иной операционной системы;
- распределенная организация операционной системы позволяет упростить работу пользователей и программистов в сетевых средах. В распределенной ОС реализованы механизмы, которые дают возможность пользователю представлять и воспринимать сеть в виде традиционного однопроцессорного компьютера. Характерными признаками распределенной организации ОС являются: наличие единой справочной службы разделяемых ресурсов, единой службы времени, использование механизма вызова удаленных процедур (RPC) для прозрачного распределения программных процедур по машинам, многонитевой обработки, позволяющей распараллеливать вычисления в рамках одной задачи и выполнять эту задачу сразу на нескольких компьютерах сети, а также наличие других распределенных служб.

5.2 Файловые системы

Файловая система - порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п.

В широком смысле понятие "файловая система" включает:

- совокупность всех файлов на диске;
- наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;
- комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

С точки зрения операционной системы, весь диск представляет собой набор кластеров (как правило, размером 512 байт и больше). Драйверы файловой системы организуют кластеры в файлы и каталоги (реально являющиеся файлами, содержащими список файлов в этом каталоге). Эти же драйверы отслеживают, какие из кластеров в настоящее время используются, какие свободны, какие помечены как неисправные.

Однако файловая система не обязательно напрямую связана с физическим носителем информации. Существуют виртуальные файловые системы, а также сетевые файловые системы, которые являются лишь способом доступа к файлам, находящимся на удалённом компьютере.

Кластер (англ. cluster) — в некоторых типах файловых систем логическая единица хранения данных в таблице размещения файлов, объединяющая группу секторов. Например, на дисках с размером секторов в 256 байт, 256-байтный кластер содержит один сектор, тогда как 2-килобайтный кластер содержит восемь секторов.

Как правило, это наименьшее место на диске, которое может быть выделено для хранения файла.

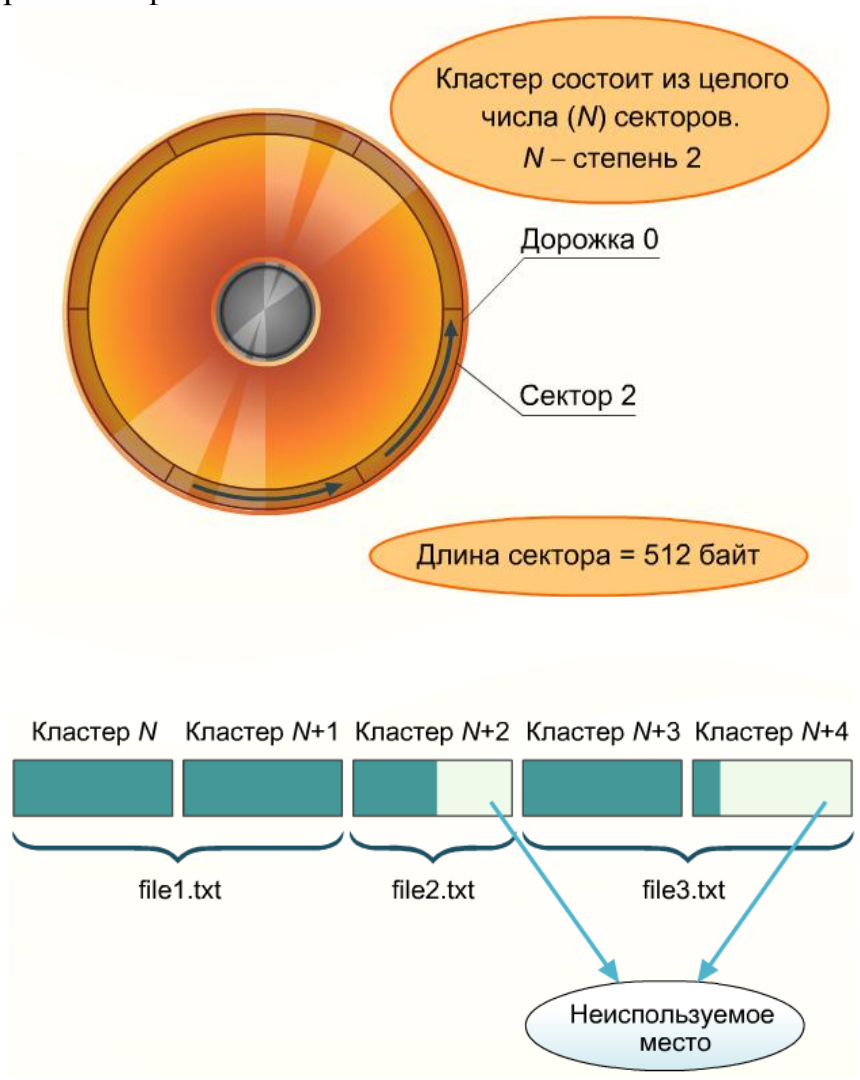


Рисунок 5.1 – Иллюстрация понятия "кластер"

Понятие кластер используется в файловых системах FAT, NTFS, а так же HFS Plus. Другие файловые системы оперируют схожими понятиями (зоны в Minix, блоки в Unix).

В некоторых файловых системах Linux (ReiserFS, Reiser4, Btrfs), BSD (FreeBSD UFS2) последний блок файла может быть поделен на подфрагменты, в которые могут быть помещены «хвосты» других файлов. В NTFS маленькие файлы могут быть записаны в Master File Table (MFT). В

файловой же системе FAT из-за примитивного алгоритма степень фрагментации постоянно растёт и требуется периодическая дефрагментация. Маленький кластер лучше подходит для маленьких файлов. Так экономнее расходуется место. Большой кластер позволяет достичь более высоких скоростей, но на мелких файлах место будет использоваться нерационально (многие сектора будут не полностью заполненными, но будут считаться занятыми).

Длинные файлы занимают несколько кластеров. Если запись производится на незаполненный диск, то кластеры, принадлежащие одному файлу, записываются подряд. Если диск переполнен, на нём может не быть цельной области, достаточной для размещения файла. Тем не менее, файл все-таки запишется, если на диске много мелких областей, суммарный размер которых достаточен для записи. В этом случае файл записывается в виде нескольких фрагментов.

Процесс разбиения файла на небольшие фрагменты при записи на диск называется **фрагментацией**. Если на диске много фрагментированных файлов, скорость чтения носителя уменьшается, поскольку поиск кластеров, в которых хранятся файлы, на жёстких дисках требует времени. На флеш-памяти, например, время поиска не зависит от расположения секторов, и практически равно нулю, поэтому для них дефрагментация не требуется.

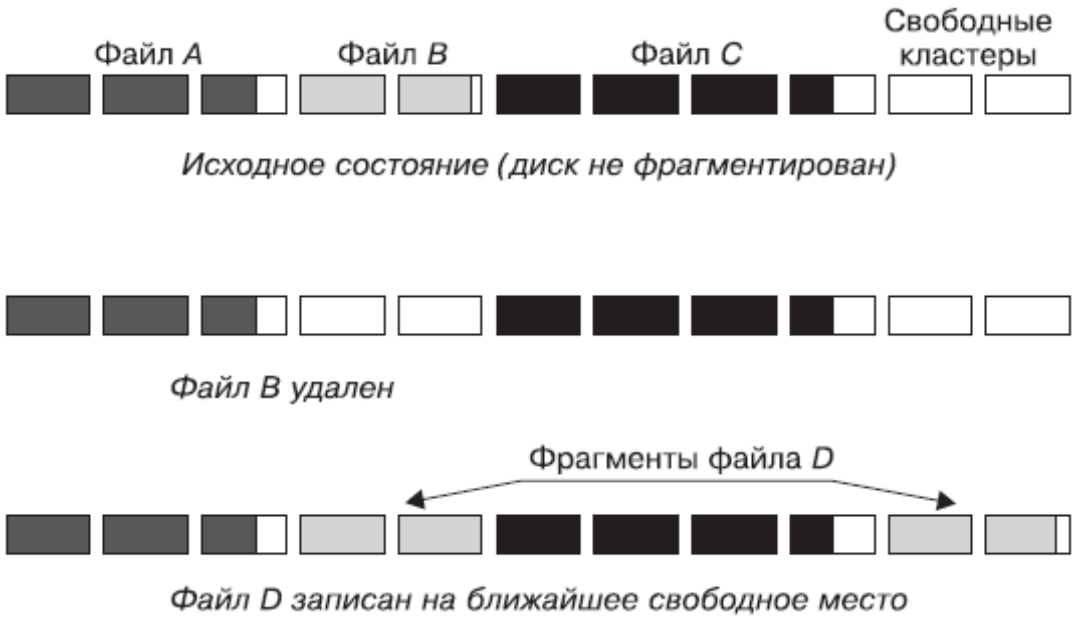


Рисунок 5.2 – Пример фрагментации

Некоторое ПО требует, чтобы определённые файлы в обязательном порядке хранились в последовательно расположенных секторах. Если в таком случае будет использоваться даже твердотельный накопитель, очевидно, дефрагментация ему всё-таки понадобится.

Дефрагментация – процесс обновления и оптимизации логической структуры раздела диска с целью обеспечения хранения файлов в непрерывной последовательности кластеров. После дефрагментации

ускоряется чтение и запись файлов, а, следовательно, и работа программ, ввиду того, что последовательные операции чтения и записи выполняются быстрее случайных обращений (например, для жесткого диска при этом не требуется перемещение головки). Другое определение дефрагментации: перераспределение файлов на диске, при котором они располагаются в непрерывных областях.

Дефрагментация чаще всего используется для таких файловых систем, как FAT/FAT32, так как в программах для работы с ними обычно не предусмотрено никаких средств для предотвращения фрагментации, и она появляется даже на почти пустом диске и небольшой нагрузке.

Помимо замедления компьютера в работе с файловыми операциями (таких как чтение и запись), фрагментация файлов негативно сказывается на «здоровье» жёсткого диска, так как заставляет постоянно перемещаться позиционирующие головки диска, которые осуществляют чтение и запись данных. Для устранения проблемы фрагментации существуют программы-дефрагментаторы, принцип работы которых заключается в «сборе» каждого файла из его фрагментов. Общим недостатком таких программ является их медленная работа — процесс дефрагментации обычно занимает очень много времени (до нескольких часов).

Имена файлов

Файлы идентифицируются именами. Пользователи дают файлам символьные имена, при этом учитываются ограничения ОС как на используемые символы, так и на длину имени. До недавнего времени эти границы были весьма узкими. Так в некогда популярной файловой системе FAT (Files Allocation Table) длина имен ограничивается известной схемой 8.3 (8 символов - собственно имя, 3 символа - расширение имени), а в ОС UNIX System V имя не может содержать более 14 символов. Однако пользователю гораздо удобнее работать с длинными именами, поскольку они позволяют дать файлу действительно мнемоническое название, по которому даже через достаточно большой промежуток времени можно будет вспомнить, что содержит этот файл. Поэтому современные файловые системы, как правило, поддерживают длинные символьные имена файлов. Например, Windows, начиная с версии NT, в своей файловой системе NTFS устанавливает, что имя файла может содержать до 255 символов, не считая завершающего нулевого символа.

При переходе к длинным именам возникает проблема совместимости с ранее созданными приложениями, использующими короткие имена. Чтобы приложения могли обращаться к файлам в соответствии с принятыми ранее соглашениями, файловая система должна уметь предоставлять эквивалентные короткие имена (псевдонимы) файлам, имеющим длинные имена. Таким образом, одной из важных задач становится проблема генерации соответствующих коротких имен.

Длинные имена поддерживаются не только новыми файловыми системами, но и новыми версиями хорошо известных файловых систем.

Например, в ОС Windows 95 использовалась файловая система VFAT (FAT32), представляющая собой существенно измененный вариант FAT. Среди многих других усовершенствований одним из главных достоинств VFAT является поддержка длинных имен. Кроме проблемы генерации эквивалентных коротких имен, при реализации нового варианта FAT важной задачей была задача хранения длинных имен при условии, что принципиально метод хранения и структура данных на диске не должны были измениться.

Обычно разные файлы могут иметь одинаковые символьные имена. В этом случае файл однозначно идентифицируется так называемым составным именем, представляющим собой последовательность символьных имен каталогов. В некоторых системах одному и тому же файлу не может быть дано несколько разных имен, а в других такое ограничение отсутствует. В последнем случае операционная система присваивает файлу дополнительно уникальное имя, так, чтобы можно было установить взаимно-однозначное соответствие между файлом и его уникальным именем. Уникальное имя представляет собой числовой идентификатор и используется программами операционной системы. Примером такого уникального имени файла является номер индексного дескриптора в системе UNIX.

Типы файлов

Файлы бывают разных типов: обычные файлы, специальные файлы, файлы-каталоги.

Обычные файлы в свою очередь подразделяются на текстовые и двоичные. Текстовые файлы состоят из строк символов, представленных в ASCII-коде. Это могут быть документы, исходные тексты программ и т.п. Текстовые файлы можно прочитать на экране и распечатать на принтере. Двоичные файлы не используют ASCII-коды, они часто имеют сложную внутреннюю структуру, например, объектный код программы или архивный файл. Все операционные системы должны уметь распознавать хотя бы один тип файлов - их собственные исполняемые файлы.

Специальные файлы - это файлы, ассоциированные с устройствами ввода-вывода, которые позволяют пользователю выполнять операции ввода-вывода, используя обычные команды записи в файл или чтения из файла. Эти команды обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления соответствующим устройством. Специальные файлы, так же как и устройства ввода-вывода, делятся на блок-ориентированные и байт-ориентированные.

Каталог - это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений (например, файлы, содержащие программы игр, или файлы, составляющие один программный пакет), а с другой стороны - это файл, содержащий системную информацию о группе файлов, его составляющих. В каталоге содержится список файлов,

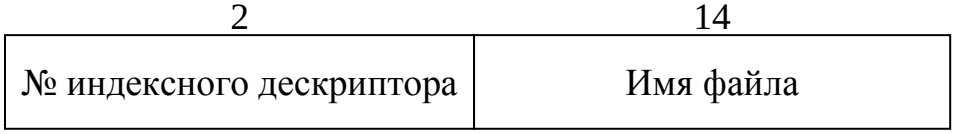
входящих в него, и устанавливается соответствие между файлами и их характеристиками (атрибутами).

В разных файловых системах могут использоваться в качестве атрибутов разные характеристики, например:

- информация о разрешенном доступе,
- пароль для доступа к файлу,
- владелец файла,
- создатель файла,
- признак "только для чтения",
- признак "скрытый файл",
- признак "системный файл",
- признак "архивный файл",
- признак "двоичный/символьный",
- признак "временный" (удалить после завершения процесса),
- признак блокировки,
- длина записи,
- указатель на ключевое поле в записи,
- длина ключа,
- времена создания, последнего доступа и последнего изменения,
- текущий размер файла,
- максимальный размер файла.

Каталоги могут непосредственно содержать значения характеристик файлов, как это сделано в файловой системе FAT, или ссылаться на таблицы, содержащие эти характеристики, как это реализовано в ОС UNIX (рисунок 5.3). Каталоги могут образовывать иерархическую структуру за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня.

а)



б)

Рисунок 5.3 – Структура каталогов:
а - структура записи каталога FAT (32 байта);
б - структура записи каталога ОС UNIX

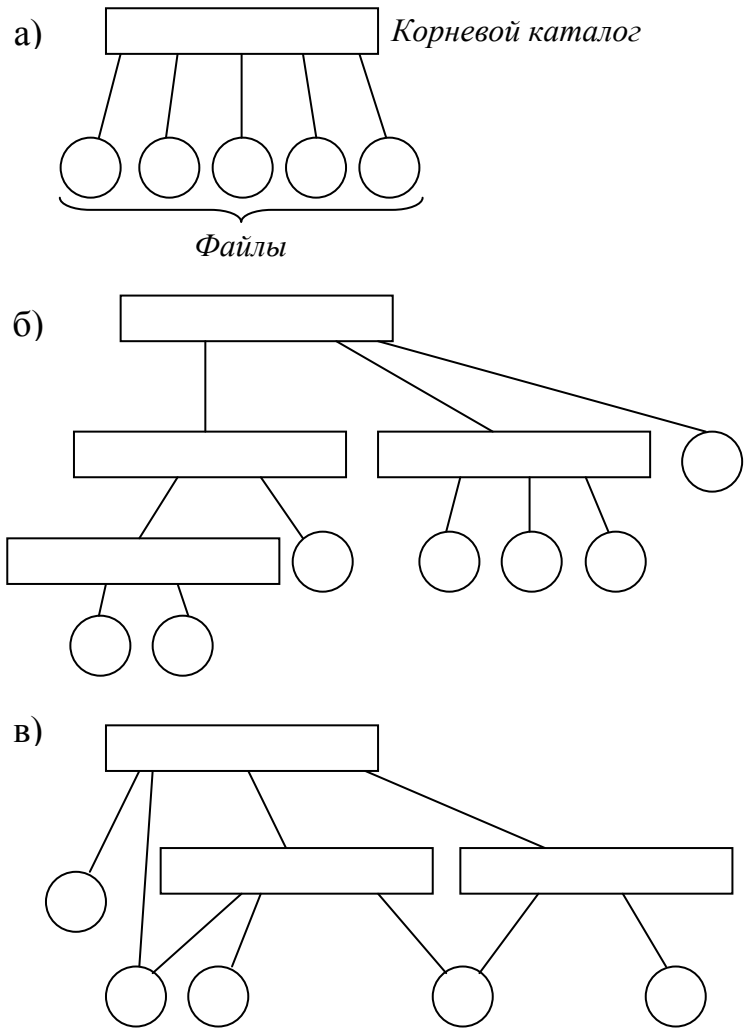


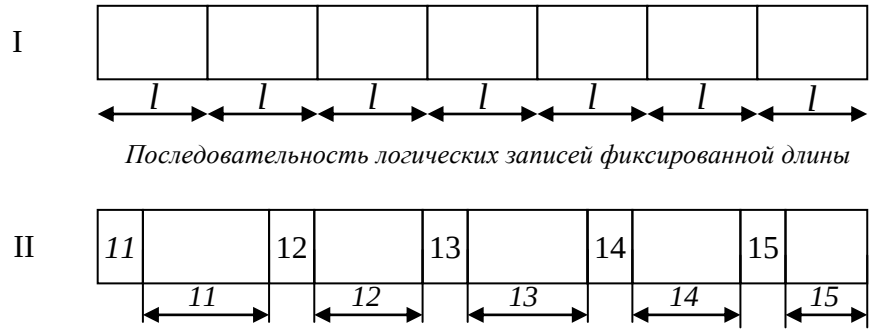
Рисунок 5.4 – Логическая организация файловой системы:
а - одноуровневая; б - иерархическая (дерево); в - иерархическая (сеть)

Иерархия каталогов может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог, и сеть - если файл может входить сразу в несколько каталогов. В Windows каталоги образуют древовидную структуру, а в UNIXe - сетевую. Как и любой другой файл, каталог имеет символьное имя и однозначно идентифицируется составным именем, содержащим цепочку символьных имен всех каталогов, через которые проходит путь от корня до данного каталога.

Логическая организация файла

Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей. Логическая запись - это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система обеспечивает программисту доступ к отдельной логической записи. На рисунке 5.5 показаны несколько схем логической

организации файла. Записи могут быть фиксированной длины или переменной длины. Записи могут быть расположены в файле последовательно (последовательная организация) или в более сложном порядке, с использованием так называемых индексных таблиц, позволяющих обеспечить быстрый доступ к отдельной логической записи (индексно-последовательная организация). Для идентификации записи может быть использовано специальное поле записи, называемое ключом. В файловых системах ОС UNIX и Windows файл имеет простейшую логическую структуру - последовательность однобайтовых записей



Индекс	1	2	3	4	5	6
Адрес	21	201	315	661	670	715

Индекс≡Ключ

Рисунок 5.5 – Способы логической организации файлов

Физическая организация и адрес файла

Физическая организация файла описывает правила расположения файла на устройстве внешней памяти, в частности, на диске. Файл состоит из физических записей - блоков. Блок - наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью. Непрерывное размещение - простейший вариант физической организации (рисунок 5.6а), при котором файлу предоставляется последовательность блоков диска, образующих единый сплошной участок дисковой памяти. Для задания адреса файла в этом случае достаточно указать только номер начального блока. Другое достоинство этого метода - простота. Но имеются и два существенных недостатка. Во-первых, во время создания файла заранее не известна его длина, а значит не известно, сколько памяти надо зарезервировать для этого файла, во-вторых, при таком порядке размещения неизбежно возникает фрагментация, и пространство на диске используется не эффективно, так как отдельные участки маленького размера (минимально 1 блок) могут остаться не используемыми.

Следующий способ физической организации - размещение в виде связанного списка блоков дисковой памяти (рисунок 5.6б).

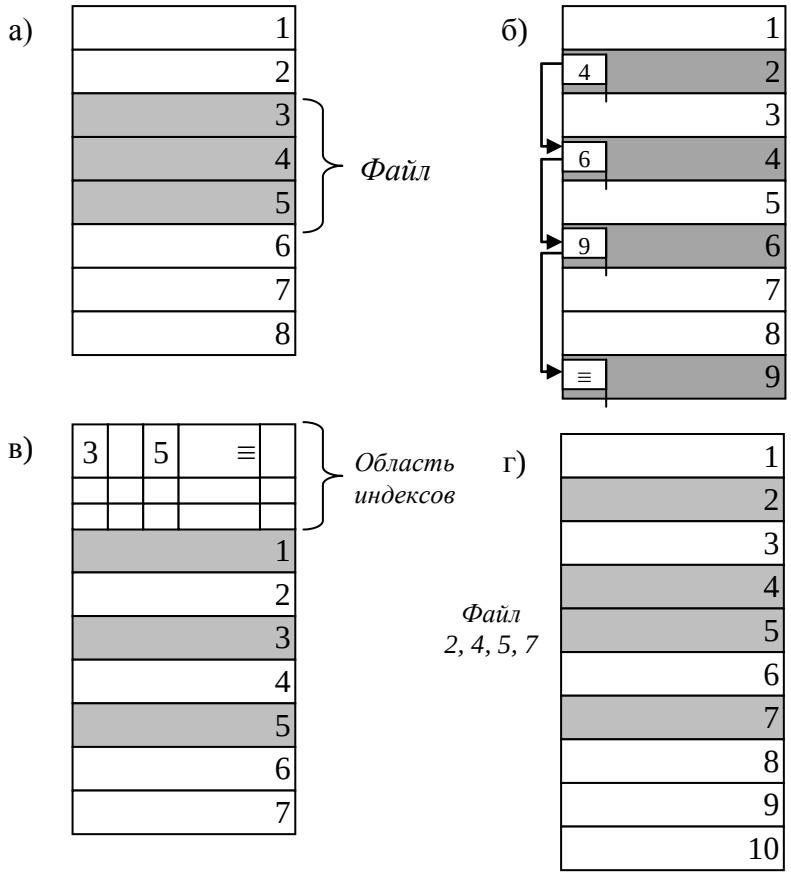


Рисунок 5.6 – Физическая организация файла:
а - непрерывное размещение; б - связанный список блоков;
в - связанный список индексов; г - перечень номеров блоков

При таком способе в начале каждого блока содержится указатель на следующий блок. В этом случае адрес файла также может быть задан одним числом - номером первого блока. В отличие от предыдущего способа, каждый блок может быть присоединен в цепочку какого-либо файла, следовательно фрагментация отсутствует. Файл может изменяться во время своего существования, наращивая число блоков. Недостатком является сложность реализации доступа к произвольно заданному месту файла: для того, чтобы прочитать пятый по порядку блок файла, необходимо последовательно прочитать четыре первых блока, прослеживая цепочку номеров блоков. Кроме того, при этом способе количество данных файла, содержащихся в одном блоке, не равно степени двойки (одно слово израсходовано на номер следующего блока), а многие программы читают данные блоками, размер которых равен степени двойки.

Популярным способом, используемым, например, в файловой системе FAT, является использование связанного списка индексов. С каждым блоком связывается некоторый элемент - индекс. Индексы располагаются в отдельной области диска (таблица FAT). Если некоторый блок распределен

некоторому файлу, то индекс этого блока содержит номер следующего блока данного файла. При такой физической организации сохраняются все достоинства предыдущего способа, но снимаются оба отмеченных недостатка: во-первых, для доступа к произвольному месту файла достаточно прочитать только блок индексов, отсчитать нужное количество блоков файла по цепочке и определить номер нужного блока, и, во-вторых, данные файла занимают блок целиком, а значит, имеют объем, равный степени двойки.

Права доступа к файлу

Определить права доступа к файлу - значит определить для каждого пользователя набор операций, которые он может применить к данному файлу. В разных файловых системах может быть определен свой список дифференцируемых операций доступа. Этот список может включать следующие операции:

- создание файла,
- уничтожение файла,
- открытие файла,
- закрытие файла,
- чтение файла,
- запись в файл,
- дополнение файла,
- поиск в файле,
- получение атрибутов файла,
- установление новых значений атрибутов,
- переименование,
- выполнение файла,
- чтение каталога,
- и другие операции с файлами и каталогами.

В самом общем случае права доступа могут быть описаны матрицей прав доступа, в которой столбцы соответствуют всем файлам системы, строки - всем пользователям, а на пересечении строк и столбцов указываются разрешенные операции (рисунок 5.7).

Имена файлов

Имена пользователей	modem.txt	win.exe	class.dbf	unix.ppt
kira	читать	выполнять	-	выполнять
genya	читать	выполнять	-	выполнять читать
nataly	читать	-	-	выполнять читать
victor	читать писать	-	создать	-

Рисунок 5.7 – Матрица прав доступа

В некоторых системах пользователи могут быть разделены на отдельные категории. Для всех пользователей одной категории определяются единые права доступа. Например, в системе UNIX все пользователи подразделяются на три категории: владельца файла, членов его группы и всех остальных.

Различают два основных подхода к определению прав доступа:

- избирательный доступ, когда для каждого файла и каждого пользователя сам владелец может определить допустимые операции;
- мандатный подход, когда система наделяет пользователя определенными правами по отношению к каждому разделяемому ресурсу (в данном случае, файлу) в зависимости от того, к какой группе пользователь отнесен.

Кэширование диска

В некоторых файловых системах запросы к внешним устройствам, в которых адресация осуществляется блоками (диски, ленты), перехватываются промежуточным программным слоем-подсистемой буферизации. Подсистема буферизации представляет собой буферный пул, располагающийся в оперативной памяти, и комплекс программ, управляющих этим пулом. Каждый буфер пула имеет размер, равный одному блоку. При поступлении запроса на чтение некоторого блока подсистема буферизации просматривает свой буферный пул и, если находит требуемый блок, то копирует его в буфер запрашивающего процесса. Операция ввода-вывода считается выполненной, хотя физического обмена с устройством не происходило. Очевиден выигрыш во времени доступа к файлу. Если же нужный блок в буферном пуле отсутствует, то он считывается с устройства и одновременно с передачей запрашивающему процессу копируется в один из буферов подсистемы буферизации. При отсутствии свободного буфера на диск вытесняется наименее используемая информация. Таким образом, подсистема буферизации работает по принципу кэш-памяти.

Общая модель файловой системы

Функционирование любой файловой системы можно представить многоуровневой моделью (рисунок 5.8), в которой каждый уровень предоставляет некоторый интерфейс (набор функций) вышележащему уровню, а сам, в свою очередь, для выполнения своей работы использует интерфейс (обращается с набором запросов) нижележащего уровня.

Задачей символьного уровня является определение по символьному имени файла его уникального имени. В файловых системах, в которых каждый файл может иметь только одно символьное имя (например, FAT32), этот уровень отсутствует, так как символьное имя, присвоенное файлу пользователем, является одновременно уникальным и может быть использовано операционной системой. В других файловых системах, в которых один и тот же файл может иметь несколько символьных имен, на данном уровне просматривается цепочка каталогов для определения

уникального имени файла. В файловой системе UNIX, например, уникальным именем является номер индексного дескриптора файла (i-node).

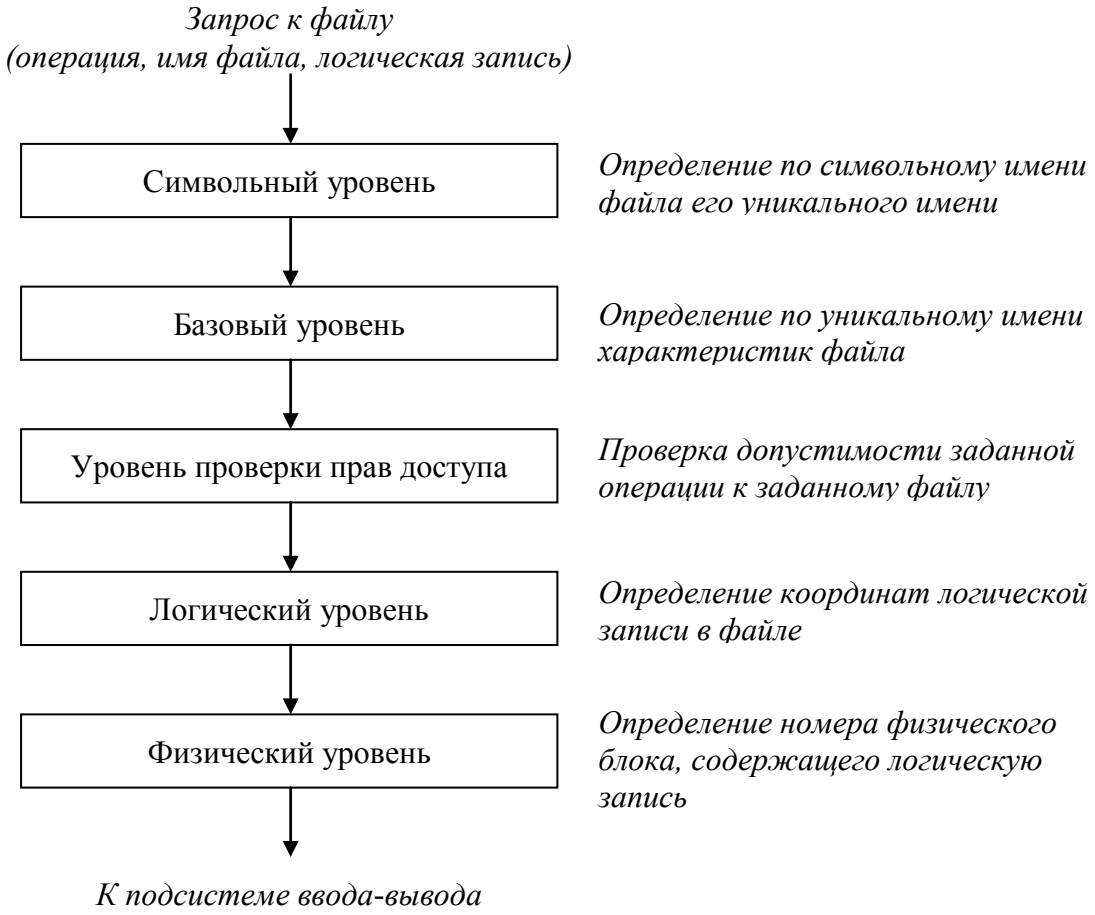


Рисунок 5.8 – Общая модель файловой системы

На следующем, базовом, уровне по уникальному имени файла определяются его характеристики: права доступа, адрес, размер и другие. Как уже было сказано, характеристики файла могут входить в состав каталога или храниться в отдельных таблицах. При открытии файла его характеристики перемещаются с диска в оперативную память, чтобы уменьшить среднее время доступа к файлу. В некоторых файловых системах (например, HPFS) при открытии файла вместе с его характеристиками в оперативную память перемещаются несколько первых блоков файла, содержащих данные.

Следующим этапом реализации запроса к файлу является проверка прав доступа к нему. Для этого сравниваются полномочия пользователя или процесса, выдавших запрос, со списком разрешенных видов доступа к данному файлу. Если запрашиваемый вид доступа разрешен, то выполнение запроса продолжается, если нет, то выдается сообщение о нарушении прав доступа.

На логическом уровне определяются координаты запрашиваемой логической записи в файле, то есть требуется определить, на каком расстоянии (в байтах) от начала файла находится требуемая логическая

запись. При этом абстрагируются от физического расположения файла, он представляется в виде непрерывной последовательности байт. Алгоритм работы данного уровня зависит от логической организации файла. Для определения координат логической записи в файле с индексно-последовательной организацией выполняется чтение таблицы индексов (ключей), в которой непосредственно указывается адрес логической записи.

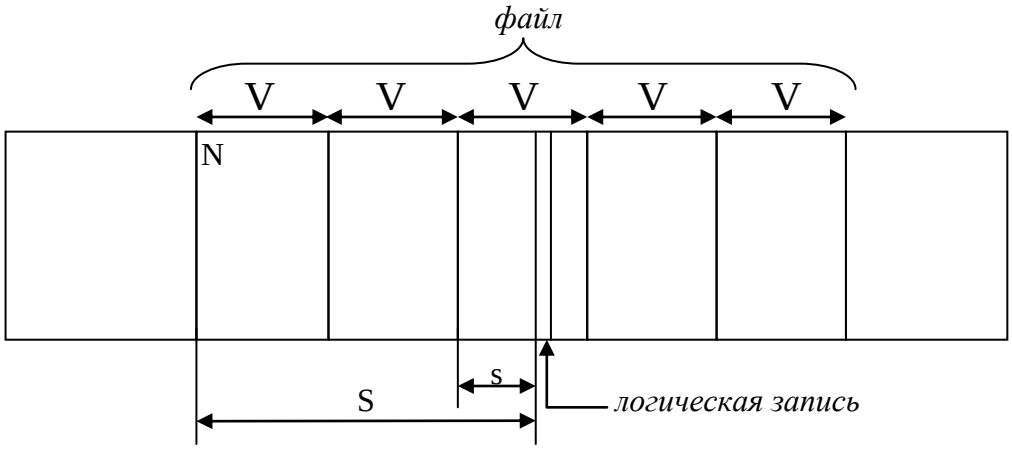


Рисунок 5.9 – Функции физического уровня файловой системы

Исходные данные:
V-размер блока
N - номер первого блока файла
S - смещение логической записи в файле.

Требуется определить на физическом уровне:
n - номер блока, содержащего требуемую логическую запись;
s - смещение логической записи в пределах блока;
$n = N + [S/V]$, где $[S/V]$ - целая часть числа S/V;
$s = R [S/V]$ - дробная часть числа S/V.

На физическом уровне файловая система определяет номер физического блока, который содержит требуемую логическую запись, и смещение логической записи в физическом блоке. Для решения этой задачи используются результаты работы логического уровня - смещение логической записи в файле, адрес файла на внешнем устройстве, а также сведения о физической организации файла, включая размер блока. Рисунок 5.9 иллюстрирует работу физического уровня для простейшей физической организации файла в виде непрерывной последовательности блоков. Подчеркнем, что задача физического уровня решается независимо от того, как был логически организован файл.

После определения номера физического блока, файловая система обращается к системе ввода-вывода для выполнения операции обмена с внешним устройством. В ответ на этот запрос в буфер файловой системы

будет передан нужный блок, в котором на основании полученного при работе физического уровня смещения выбирается требуемая логическая запись.

Современные архитектуры файловых систем

Разработчики новых операционных систем стремятся обеспечить пользователя возможностью работать сразу с несколькими файловыми системами. В новом понимании файловая система состоит из многих составляющих, в число которых входят и файловые системы в традиционном понимании.

Новая файловая система имеет многоуровневую структуру (рисунок 5.10), на верхнем уровне которой располагается так называемый переключатель файловых систем. Он обеспечивает интерфейс между запросами приложения и конкретной файловой системой, к которой обращается это приложение. Переключатель файловых систем преобразует запросы в формат, воспринимаемый следующим уровнем - уровнем файловых систем.

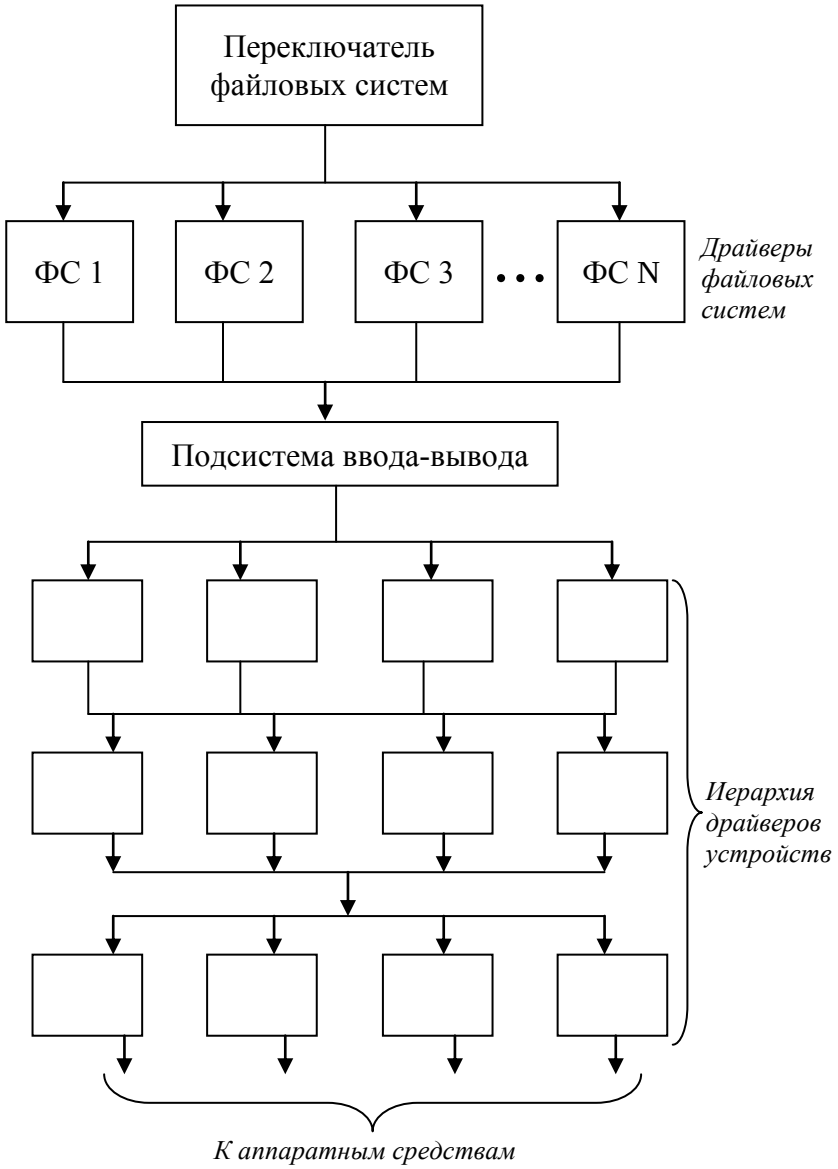


Рисунок 5.10 – Архитектура современной файловой системы

Каждый компонент уровня файловых систем выполнен в виде драйвера соответствующей файловой системы и поддерживает определенную организацию файловой системы. Переключатель является единственным модулем, который может обращаться к драйверу файловой системы. Приложение не может обращаться к нему напрямую. Драйвер файловой системы может быть написан в виде реентерабельного кода, что позволяет сразу нескольким приложениям выполнять операции с файлами. Каждый драйвер файловой системы в процессе собственной инициализации регистрируется у переключателя, передавая ему таблицу точек входа, которые будут использоваться при последующих обращениях к файловой системе.

Для выполнения своих функций драйверы файловых систем обращаются к подсистеме ввода-вывода, образующей следующий слой файловой системы новой архитектуры. Подсистема ввода-вывода - это составная часть файловой системы, которая отвечает за загрузку, инициализацию и управление всеми модулями низших уровней файловой системы. Обычно эти модули представляют собой драйверы портов, которые непосредственно занимаются работой с аппаратными средствами. Кроме этого подсистема ввода-вывода обеспечивает некоторый сервис драйверам файловой системы, что позволяет им осуществлять запросы к конкретным устройствам. Подсистема ввода-вывода должна постоянно присутствовать в памяти и организовывать совместную работу иерархии драйверов устройств. В эту иерархию могут входить драйверы устройств определенного типа (драйверы жестких дисков или накопителей на лентах), драйверы, поддерживаемые поставщиками (такие драйверы перехватывают запросы к блочным устройствам и могут частично изменить поведение существующего драйвера этого устройства, например, зашифровать данные), драйверы портов, которые управляют конкретными адаптерами.

Большое число уровней архитектуры файловой системы обеспечивает авторам драйверов устройств большую гибкость - драйвер может получить управление на любом этапе выполнения запроса - от вызова приложением функции, которая занимается работой с файлами, до того момента, когда работающий на самом низком уровне драйвер устройства начинает просматривать регистры контроллера. Многоуровневый механизм работы файловой системы реализован посредством цепочек вызова.

В ходе инициализации драйвер устройства может добавить себя к цепочке вызова некоторого устройства, определив при этом уровень последующего обращения. Подсистема ввода-вывода помещает адрес целевой функции в цепочку вызова устройства, используя заданный уровень для того, чтобы должным образом упорядочить цепочку. По мере выполнения запроса, подсистема ввода-вывода последовательно вызывает все функции, ранее помещенные в цепочку вызова.

Внесенная в цепочку вызова процедура драйвера может решить передать запрос дальше - в измененном или в неизмененном виде - на

следующий уровень, или, если это возможно, процедура может удовлетворить запрос, не передавая его дальше по цепочке.

NTFS (New Technology File System — «файловая система новой технологии») — стандартная файловая система для семейства современных версий операционных систем Microsoft Windows (начиная от NT).

NTFS заменила использовавшуюся в MS-DOS и Microsoft Windows файловую систему FAT. NTFS поддерживает систему метаданных и использует специализированные структуры данных для хранения информации о файлах для улучшения производительности, надёжности и эффективности использования дискового пространства. NTFS хранит информацию о файлах в главной файловой таблице — Master File Table (MFT). NTFS имеет встроенные возможности разграничивать доступ к данным для различных пользователей и групп пользователей (списки контроля доступа — Access Control Lists (ACL)), а также назначать квоты (ограничения на максимальный объём дискового пространства, занимаемый теми или иными пользователями). NTFS использует систему журналирования USN для повышения надёжности файловой системы.

NTFS разработана на основе файловой системы HPFS (High Performance File System — высокопроизводительная файловая система), создававшейся Microsoft совместно с IBM для операционной системы OS/2. Но, получив такие несомненно полезные новшества, как квотирование, журналируемость, разграничение доступа и аудит, в значительной степени утратила присущую прародительнице (HPFS) весьма высокую производительность файловых операций.

Extended File System (расширенная файловая система), сокращённо ext или extfs — первая файловая система, разработанная специально для ОС на ядре Linux. Представлена в апреле 1992 г.для ядра Linux 0.96c. Ext является первой версией расширенной файловой системы.

Second Extended File System (вторая расширенная файловая система), сокращённо ext2. По скорости и производительности работы она может служить эталоном в тестах производительности файловых систем. Главный недостаток ext2 (и одна из причин демонстрации столь высокой производительности) заключается в том, что она не является журналируемой файловой системой. Он был устранён в файловой системе ext3 — следующей версии Extended File System, полностью совместимой с ext2.

Файловая система ext2 по-прежнему используется на флеш-картах и твердотельных накопителях (SSD), так как отсутствие журналирования является преимуществом при работе с накопителями, имеющими ограничение на количество циклов записи.

Third Extended File System (третья версия расширенной файловой системы), сокращённо ext3 или ext3fs — журналируемая файловая система, используемая в операционных системах на ядре Linux, является файловой системой по умолчанию во многих дистрибутивах.

Основное отличие от ext2 состоит в том, что ext3 журналируема, то есть в ней предусмотрена запись некоторых данных, позволяющих восстановить файловую систему при сбоях в работе компьютера.

Fourth Extended File System (четвёртая версия расширенной файловой системы), сокр. ext4, или ext4fs — журналируемая файловая система, используемая в ОС с ядром Linux. Основана на файловой системе ext3, которая является файловой системой по умолчанию во многих дистрибутивах GNU/Linux.

Основной особенностью стало увеличение максимального объёма одного раздела диска до 1 эксбибайта (2^{60} байт) при размере блока 4Kb, и увеличение размера одного файла до 16 тебибайт. Кроме того, в ext4 представлен механизм пространственной (extent) записи файлов (новая информация добавляется в конец заранее выделенной по соседству области файла), уменьшающий фрагментацию и повышающий производительность.

5.3 Принципы организации ЭВМ

Обработка информации и представление результатов обработки в удобном для человека виде производится с помощью вычислительных средств. Научно-технический прогресс привел к созданию разнообразных вычислительных средств: электронных вычислительных машин (ЭВМ), вычислительных систем, вычислительных сетей. Они различаются структурной организацией и функциональными возможностями.

Дать определение такому явлению, как ЭВМ, представляется сложным. Достаточно сказать, что само по себе название ЭВМ, т.е. электронные вычислительные машины, не отражает полностью сущность концепции. Слово «электронные» подразумевало электронные лампы в качестве элементной базы, современные ЭВМ правильнее следовало бы называть микроэлектронными. Слово «вычислительный» подразумевает, что устройство предназначено для проведения вычислений, однако анализ программ показывает, что современные ЭВМ не более 10 – 15% времени тратят на чисто вычислительную работу - сложение, вычитание, умножение и т.д. Основное время затрачивается на выполнение операций пересылки данных, сравнения, ввода-вывода и т.д. То же самое относится и к англоязычному термину «компьютер», т.е. «вычислитель». К понятию ЭВМ можно подходить с нескольких точек зрения.

Представляется разумным определить ЭВМ с точки зрения ее функционирования. Целесообразно описать минимальный набор устройств, который входит в состав любой ЭВМ, и тем самым определить состав минимальной ЭВМ, а также сформулировать принципы работы отдельных блоков ЭВМ и принципы организации ЭВМ как системы, состоящей из взаимосвязанных функциональных блоков.

Если же рассматривать ЭВМ как ядро некоторой информационно-вычислительной системы, может оказаться полезным показать

информационную модель ЭВМ – определить ее в виде совокупности блоков переработки информации и множества информационных потоков между этими блоками.

Принципы фон-Неймана. Большинство современных ЭВМ строится на базе принципов, сформулированных американским ученым, одним из «отцов» кибернетики Дж. фон Нейманом. Впервые эти принципы были опубликованы фон Нейманом в 1945 г. в его предложениях по машине EDVAC. Эта ЭВМ была одной из первых машин с хранимой программой, т.е. с программой, запомненной в памяти машины, а не считываемой с перфокарты или другого подобного устройства. В целом эти принципы сводятся к следующему:

1) Основными блоками фон-неймановской машины являются блок управления, арифметико-логическое устройство, память и устройство ввода-вывода (рисунок 5.11).

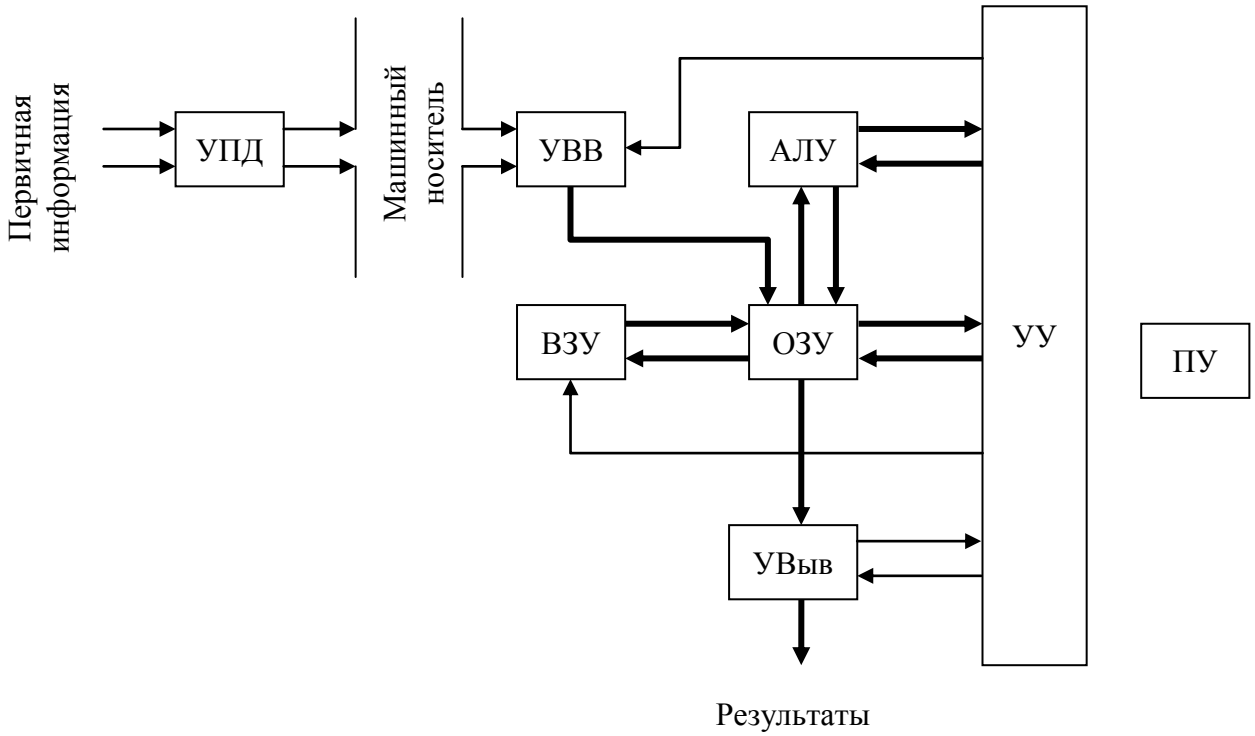


Рисунок 5.11 – Обобщенная структурная схема ЭВМ:
УПД – устройство подготовки данных, УВВ – устройство ввода информации, ОЗУ – оперативное запоминающее устройство, ВЗУ – внешнее запоминающее устройство, АЛУ – арифметико-логическое устройство, УУ – устройство управления, ПУ – пульт управления, Увыв – устройство вывода

2) Информация кодируется в двоичной форме и разделяется на единицы, называемые словами.

3) Алгоритм представляется в форме последовательности управляющих слов, которые определяют смысл операции. Эти управляющие слова

называются командами. Совокупность команд, представляющая алгоритм, называется *программой*.

4) Программы и данные хранятся в одной и той же памяти. Разнотипные слова различаются по способу использования, но не по способу кодирования.

5) Устройство управления и арифметическое устройство обычно объединяются в одно, называемое центральным процессором. Они определяют действия, подлежащие выполнению, путем считывания команд из оперативной памяти. Обработка информации, предписанная алгоритмом, сводится к последовательному выполнению команд в порядке, однозначно определяемом программой.

Принципы фон-Неймана практически можно реализовать множеством различных способов. Перед тем как описать принципы функционирования ЭВМ, введем несколько определений. *Архитектура ЭВМ* - абстрактное определение машины в терминах основных функциональных модулей, языка, структур данных. Архитектура не определяет особенности реализации аппаратной части ЭВМ, времени выполнения команд, степени параллелизма, ширины шин и других аналогичных характеристик. Архитектура отображает аспекты структуры ЭВМ, которые являются видимыми для пользователя: систему команд, режимы адресации, форматы данных, набор программно-доступных регистров. Одним словом, термин «архитектура» используется для описания возможностей, предоставляемых ЭВМ. Весьма часто употребляется термин конфигурация ЭВМ, под которым понимается компоновка вычислительного устройства с четким определением характера, количества, взаимосвязей и основных характеристик его функциональных элементов. Термин «организация ЭВМ» определяет, как реализованы возможности ЭВМ.

Команда - совокупность сведений, необходимых процессору для выполнения определенного действия при выполнении программы. Команда состоит из *кода операции*, содержащего указание на операцию, которую необходимо выполнить, и нескольких *адресных полей*, содержащих указание на места расположения операндов команды. Способ вычисления адреса по информации, содержащейся в адресном поле команды, называется *режимом адресации*. Множество команд, реализованных в данной ЭВМ образует ее систему команд.

Гарвардская архитектура. Типичные операции (сложение и умножение) требуют от любого вычислительного устройства нескольких действий:

1.выборку двух операндов,

2.выбор инструкции и её выполнение,

3.и, наконец, сохранение результата.

Идея такой архитектуры заключалась в физическом разделении линий передачи команд и данных. В первом компьютере Эйкена «Марк I» для

хранения инструкций использовалась перфорированная лента, а для работы с данными — электромеханические регистры. Это позволяло одновременно пересылать и обрабатывать команды и данные, благодаря чему значительно повышалось общее быстродействие компьютера.

В Гарвардской архитектуре характеристики устройств памяти для инструкций и памяти для данных не требуется иметь общими. В частности, ширина слова, тайминги, технология реализации и структура адресов памяти могут различаться. В некоторых системах инструкции могут храниться в памяти только для чтения, в то время как для сохранения данных обычно требуется память с возможностью чтения и записи. В некоторых системах требуется значительно больше памяти для инструкций, чем памяти для данных, поскольку данные обычно могут подгружатся с внешней или более медленной памяти. Такая потребность увеличивает битность (ширину) шины адреса памяти инструкций по сравнению с шиной адреса памяти данных.

Отличие от архитектуры фон Неймана

В чистой архитектуре фон Неймана процессор одномоментно может либо читать инструкцию, либо читать/записывать единицу данных из/в памяти. То и другое не может происходить одновременно, поскольку инструкции и данные используют одну и ту же системную шину, тогда как в компьютере с использованием гарвардской архитектуры процессор может читать инструкции и выполнять доступ к памяти данных в то же самое время, даже без кэш-памяти. Таким образом, компьютер с гарвардской архитектурой может быть быстрее (при определенной сложности схемы), поскольку доставка инструкций и доступ к данным не претендуют на один и тот же канал памяти.

Также машина гарвардской архитектуры имеет различные адресные пространства для команд и данных. Так, нулевой адрес инструкций — это не то же самое, что и нулевой адрес данных. Нулевой адрес инструкций может определятся двадцатичетырехбитным значением, в то время как нулевой адрес данных может выглядеть как восьмибитный байт, который не являются частью этого двадцатичетырехбитного значения.

Модифицированная гарвардская архитектура

Соответствующая схема реализации доступа к памяти имеет один очевидный недостаток — высокую стоимость. При разделении каналов передачи команд и данных на кристалле процессора последний должен иметь почти вдвое больше выводов, так как шина адреса и шина данных составляют основную часть выводов микропроцессора. Способом решения этой проблемы стала идея использовать общие шину данных и шину адреса для всех внешних данных, а внутри процессора использовать шину данных, шину команд и две шины адреса. Такую концепцию стали называть модифицированной Гарвардской архитектурой.

Такой подход применяется в современных сигнальных процессорах. Ещё дальше по пути уменьшения стоимости пошли при создании однокристалльных ЭВМ — микроконтроллеров. В них одна шина команд и данных применяется и внутри кристалла.

Разделение шин в модифицированной Гарвардской структуре осуществляется при помощи раздельных управляющих сигналов: чтения, записи или выбора области памяти.

Расширенная гарвардская архитектура

Часто требуется выбрать три составляющие : два операнда и инструкцию (в алгоритмах цифровой обработки сигналов это наиболее распространенная задача в БПФ и КИХ, БИХ фильтрах). Для этого существует кэш-память. В ней может храниться инструкция — следовательно, обе шины остаются свободными и появляется возможность передать два операнда одновременно. Использование кэш-памяти вместе с разделёнными шинами получило название «Super Harvard Architecture» («SHARC») — расширенная Гарвардская архитектура.

Примером могут служить процессоры «Analog Devices»: ADSP-21xx — модифицированная Гарвардская Архитектура, ADSP-21xxx(SHARC) — расширенная Гарвардская Архитектура.

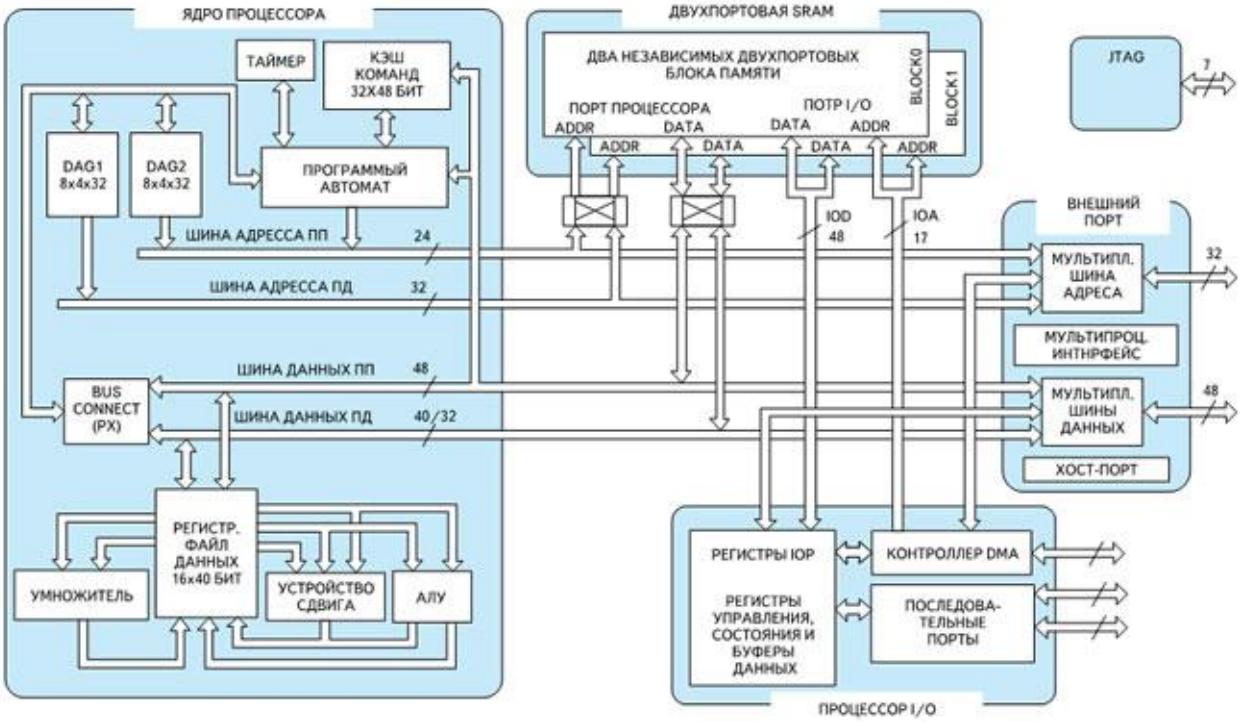


Рисунок 5.12 – Архитектура процессора ADSP-21061

Гибридные модификации с архитектурой фон Неймана

Существуют гибридные архитектуры, сочетающие достоинства как Гарвардской так и фон Неймановской архитектур. Современные CISC-процессоры обладают раздельной кэш-памятью 1-го уровня для инструкций и данных, что позволяет им за один рабочий такт получать одновременно и команду, и данные для её выполнения. То есть процессорное ядро, формально, является гарвардским, но программно оно фон Неймановское, что упрощает написание программ. Обычно в данных процессорах одна шина используется и для передачи команд, и для передачи данных, что упрощает конструкцию системы. Современные варианты таких процессоров могут иногда содержать встроенные контроллеры сразу нескольких разнотипных

шин для работы с различными типами памяти — например, DDR RAM и Flash. Тем не менее, и в этом случае шины, как правило, используются и для передачи команд, и для передачи данных без разделения, что делает данные процессоры ещё более близкими к фон Неймановской архитектуре при сохранении плюсов Гарвардской архитектуры.

Функционирование ЭВМ с шинной организацией

Шинная организация является простейшей формой организации ЭВМ. В соответствии с приведенными выше принципами фон-Неймана подобная ЭВМ имеет в своем составе следующие функциональные блоки (рисунок 5.13).

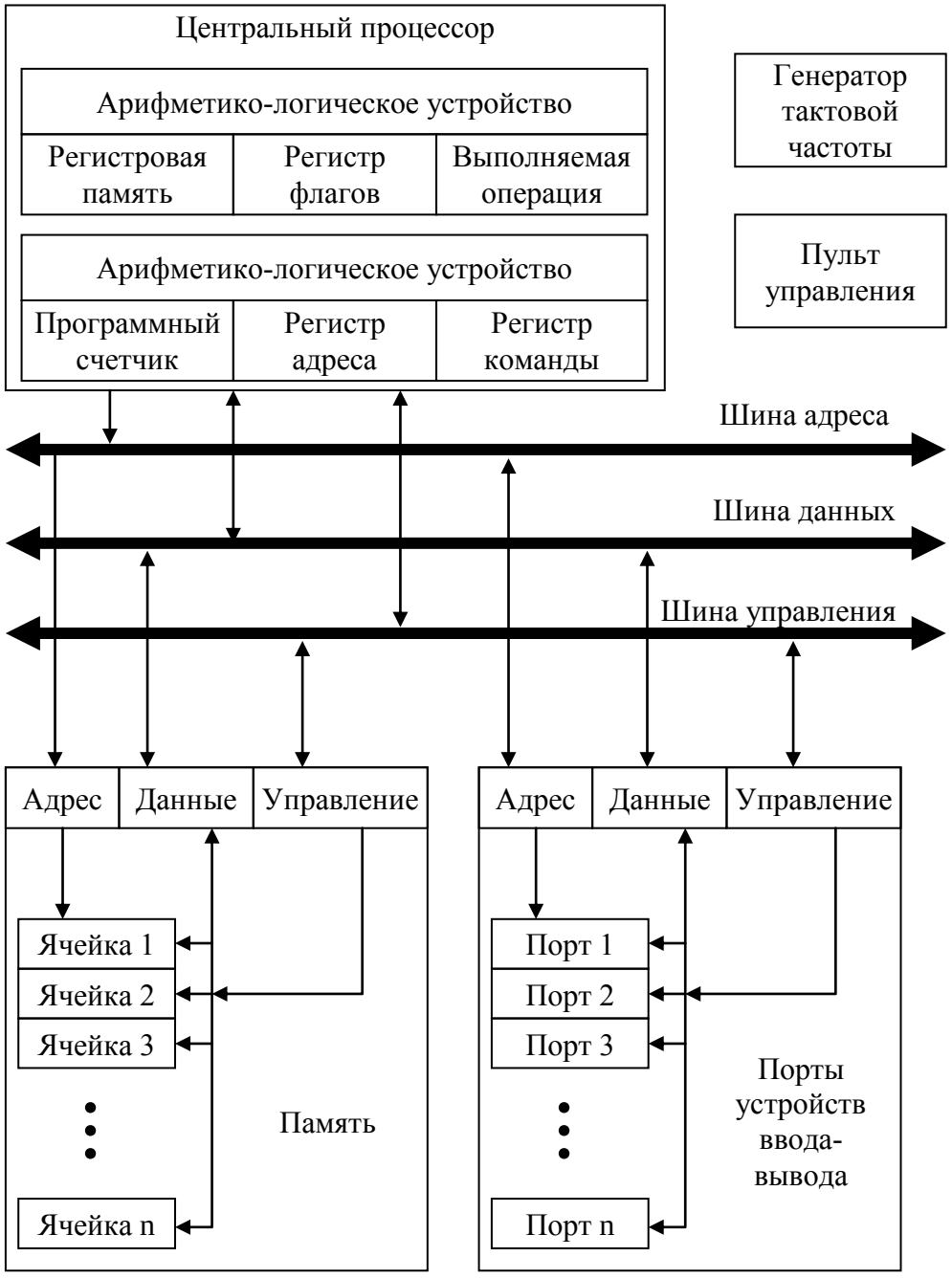


Рисунок 5.13 – Упрощенная схема ЭВМ с шинной организацией

Центральный процессор (ЦП) - функциональная часть ЭВМ, выполняющая основные операции по обработке данных и управлению работой других блоков. Это наиболее сложный компонент ЭВМ как с точки зрения электроники, так и с точки зрения функциональных возможностей. Центральный процессор состоит из следующих взаимосвязанных составных элементов: арифметико-логического устройства, устройства управления и регистров.

Арифметико-логическое устройство (АЛУ) выполняет основную работу по переработке информации, хранимой в оперативной памяти. В нем выполняются арифметические и логические операции. Кроме того, АЛУ вырабатывает управляющие сигналы, позволяющие ЭВМ автоматически выбирать путь вычислительного процесса в зависимости от получаемых результатов. Операции выполняются с помощью электронных схем, каждая из которых состоит из сотен тысяч элементов. Микросхемы имеют высокую плотность и быстродействие. На современном технологическом уровне все АЛУ можно разместить на одном кристалле полупроводникового элемента размером значительно меньше головки спички.

Арифметико-логическое устройство формирует по двум входным переменным одну выходную, выполняя заданную функцию (сложение, вычитание, сдвиг и т.д.). Выполняемая функция определяется микрокомандой, получаемой от устройства управления. АЛУ содержит в своем составе устройство, хранящее характеристику результата выполнения операции над данными и называемое *флаговым регистром*. Отметим пока, что отдельные разряды этого регистра указывают на равенство результата операции нулю, знак результата операции (+ или -), правильность выполнения операции (наличие переноса за пределы разрядной сетки или переполнения). Программный анализ флагов позволяет производить операции ветвления программы в зависимости от конкретных значений данных.

Кроме того, в АЛУ имеется набор программно-доступных быстродействующих ячеек памяти, которые называются *регистрами процессора*.

Регистры составляют основу архитектуры процессора. Среди обязательного набора регистров можно отметить следующие. *Регистр данных* - служит для временного хранения промежуточных результатов при выполнении операций. *Регистр-аккумулятор* - регистр временного хранения, который используется в процессе вычислений (например, в нем формируется результат выполнения команды умножения). *Регистр-указатель стека* - используется при операциях со стеком, т.е. такой структурой данных, которая работает по принципу: последним вошел - первым вышел, т.е. последнее записанное в него значение извлекается из него первым. Пока отметим только, что стеки используются для организации подпрограмм. *Индексные, указательные и базовые регистры* используются для хранения и вычисления адресов операндов в памяти. *Регистры-счетчики* используются для организации циклических участков в программах. *Регистры общего*

назначения, имеющиеся во многих ЭВМ, могут использоваться для любых целей. Точное назначение такого регистра определяет программист при написании программы. Они могут использоваться для временного хранения данных, в качестве аккумуляторов, а также в качестве индексных, базовых, указательных регистров. Количество регистров и связей между ними оказывает существенное влияние на сложность и стоимость процессора. Однако, с другой стороны, наличие большого количества регистров с богатым набором возможностей упрощает программирование и повышает гибкость программного обеспечения. Кроме перечисленных регистров в состав АЛУ могут входить *внутренние системные регистры*, не доступные программно и используемые во время внутренних пересылок информации при выполнении команд.

Устройство управления (УУ) - часть центрального процессора. Оно вырабатывает распределенную во времени и пространстве последовательность внутренних и внешних управляющих сигналов, обеспечивающих выборку и выполнение команд. На этапе цикла выборки команды УУ интерпретирует команду, выбранную из программной памяти. На этапе выполнения команды в соответствии с типом реализуемой операции УУ формирует требуемый набор команд низкого уровня для арифметико-логического устройства и других устройств. Эти команды задают последовательность простейших низкоуровневых операций, таких, как пересылка данных, сдвиг данных, установка и анализ признаков, запоминание результатов и др. Такие элементарные низкоуровневые операции называют *микрооперациями*, а команды, формируемые устройством управления, называются *микрокомандами*. Последовательность микрокоманд, соответствующая одной команде, называется *микропрограммой*.

В простейшем случае УУ имеет в своем составе три устройства - *регистр команды*, который содержит код команды во время ее выполнения, *программный счетчик*, в котором содержится адрес очередной подлежащей выполнению команды, *регистр адреса*, в котором вычисляются адреса операндов, находящихся в памяти. Для связи пользователя с ЭВМ предусмотрен пульт управления, который позволяет выполнять такие действия, как сброс ЭВМ в начальное состояние, просмотр регистра или ячейки памяти, запись адреса в программный счетчик, пошаговое выполнение программы при ее отладке и т.д.

Память (ПАМ) - устройство, предназначенное для запоминания, хранения и выборки программ и данных. Память состоит из конечного числа ячеек, каждая из которых имеет свой уникальный номер или адрес. Доступ к ячейке осуществляется указанием ее адреса. Память способна выполнять два вида операций над данными - чтение с сохранением содержимого и запись нового значения со стиранием предыдущего. Как уже говорилось выше, каждая ячейка памяти может использоваться для хранения либо порции данных, либо команды. В большинстве современных ЭВМ минимально адресуемым элементом памяти является *байт* - поле из 8 бит. Совокупность

битов, которые арифметико-логическое устройство может одновременно поместить в регистр или обработать, называют обычно *машинным словом*.

Оперативная память (ОП) - функциональный блок, хранящий информацию для УУ (команды) и АЛУ (данные). Задачи, решаемые с помощью ЭВМ, требуют хранения в памяти различного количества информации, зависящего от сложности реализуемого алгоритма, количества исходных данных и т.п. Поэтому память должна вмещать достаточно большое количество информации, т.е. должна иметь большую емкость. С другой стороны, память должна обладать достаточным быстродействием, соответствующим быстродействию других устройств ЭВМ. Чем больше емкость памяти, тем медленнее к ней доступ, так как время доступа (т.е. быстродействие) определяется временем, необходимым для выборки из памяти или записи в нее информации. Поэтому в ЭВМ существует несколько запоминающих устройств, различающихся емкостью и быстродействием.

Оперативная память собирается на полупроводниковых микросхемах и состоит из отдельных ячеек.

Периферийные устройства (ПУ). В их число входят устройства двух типов: устройства внешней памяти, предназначенные для долговременного хранения данных большого объема и программ, и коммуникационные устройства, предназначенные для связи ЭВМ с внешним миром (с пользователем, другими ЭВМ и т.д.). Обмен данными с внешним устройством осуществляется через *порты ввода-вывода*. Порт (в переводе с англ, port - ворота, дверь, отверстие) - это абстрактное понятие, на самом деле несуществующее. По аналогии с ячейками памяти порты можно рассматривать как ячейки, через которые можно записать в ПУ, или, наоборот, прочитать из него. Так же как и ячейки памяти, порты имеют уникальные номера - адреса портов ввода-вывода.

Система шин. Объединение функциональных блоков в ЭВМ осуществляется посредством следующей системы шин: *шины данных*, по которой осуществляется обмен информацией между блоками ЭВМ, *шины адреса*, используемой для передачи адресов (номеров ячеек памяти или портов ввода-вывода, к которым производится обращение), и *шины управления* для передачи управляющих сигналов. Совокупность этих трех шин называют *системной шиной, системной магистралью* или *системным интерфейсом*. Состав и назначение шин и правило их использования, виды передаваемых по шине сигналов и другие характеристики шины могут существенно различаться у разных видов ЭВМ. Однако есть принципиально общие закономерности в организации шин. Шина состоит из отдельных проводников (*линий*). Сигналы по линиям шины могут передаваться либо импульсами (наличие импульса соответствует логической 1, а отсутствие импульса - 0), либо уровнем напряжения (например, высокий уровень - логическая 1 , низкий - 0). *Шириной шины* называется количество линий (проводников), входящих в состав шины. Ширина шины адреса определяет размер адресного пространства ЭВМ. Если, например, количество линий адреса, используемых для адресации памяти, равно 20, то общее количество

адресуемых ячеек памяти составит 2^{20}, т.е. примерно, один миллион ячеек (точнее, 1 048 576 ячеек).

Функционирование ЭВМ с шинной структурой можно описать следующим обобщенным алгоритмом (рисунок 5.14):

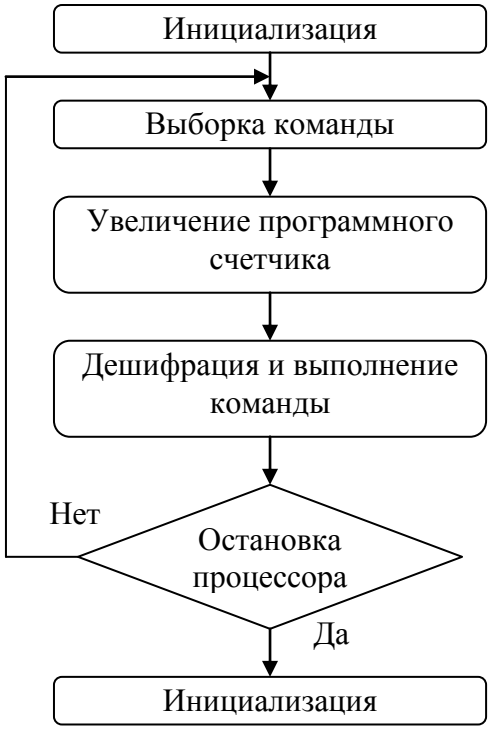


Рисунок 5.14 – Обобщенный алгоритм функционирования фон-неймановской ЭВМ

1. Инициализация. После включения ЭВМ или операции сброса в регистры центрального процессора заносятся некоторые начальные значения. Обычно в процессе инициализации в память ЭВМ помещается программа, называемая первичным загрузчиком. Основное назначение первичного загрузчика - загрузить в память с устройства внешней памяти операционную систему. Эта программа может быть размещена в энергонезависимом устройстве памяти или автоматически считываться с некоторого устройства внешней памяти. Мы не будем здесь подробно останавливаться на механизмах загрузки операционной системы, тем более что они могут существенно различаться для разных типов ЭВМ. Пока будем полагать, что в памяти некоторым образом оказалась первая из подлежащих выполнению программ. Программному счетчику присваивается начальное значение, равное адресу первой команды программы, указанной выше.

2. Центральный процессор производит операцию считывания команды из памяти. В качестве адреса ячейки памяти используется содержимое программного счетчика.

3. Содержимое считанной ячейки памяти интерпретируется процессором как команда и помещается в регистр команды. Устройство управления поступает к интерпретации посчитанной команды. По полю кода

операции из первого слова команды устройство управления определяет ее длину и, если это необходимо, организует дополнительные операции считывания, пока вся команда полностью не будет прочитана процессором. Вычисленная длина команды прибавляется к исходному содержимому программного счетчика, и когда команда полностью прочитана, программный счетчик будет хранить адрес следующей команды.

4. По адресным полям команды устройство управления определяет, имеет ли команда операнды в памяти. Если это так, то на основе указанных в адресных полях режимов адресации вычисляются адреса операндов и производятся операции чтения памяти для считывания операндов.

5. Устройство управления и арифметико-логическое устройство выполняют операцию, указанную в поле кода операции команды. Во флаговом регистре процессора запоминаются признаки результата операции (равно нулю или нет, знак результата, наличие переполнения и т.д.).

6. Если это необходимо, устройство управления выполняет операцию записи для того, чтобы поместить результат выполнения команды в память.

7. Если последняя команда не была командой ОСТАНОВИТЬ ПРОЦЕССОР, то описанная последовательность действий повторяется, начиная с шага 1. Описанная последовательность действий центрального процессора с шага 1 до шага 6 называется циклом процессора.

Большинство мини- и микроЭВМ имеют шинную организацию и их поведение описывается приведенным выше алгоритмом. В различных конкретных ЭВМ реализация этого алгоритма может несколько отличаться. Так, например, по-разному может осуществляться синхронизация, процессор может считывать из памяти не одну команду, а сразу несколько и хранить их в специальной очереди команд. Часто используемые программой команды и данные могут храниться не в основной памяти ЭВМ, а в быстродействующей буферной памяти и т.д. Таким образом, функционирование любой фон-неймановской ЭВМ описывается алгоритмом, близким к приведенному выше, и представляет собой последовательность достаточно простых действий.

Функционирование ЭВМ с канальной организацией

В основе этого типа организации ЭВМ лежит множественность каналов связи между устройствами и функциональная специализация узлов. Упрощенная схема организации ЭВМ с каналами приведена на рисунке 5.15. Сравним схему ЭВМ с каналами и описанную выше схему ЭВМ с шинной организацией.

Все фон-неймановские ЭВМ очень похожи друг на друга и алгоритм функционирования центрального процессора по сути ничем не отличается от описанного выше.

Помимо уже знакомого набора устройств (центральный процессор, память, устройства ввода-вывода) в состав ЭВМ с канальной организацией входят устройства, называемые каналами. *Канал* - это специализированный

процессор, осуществляющий всю работу по управлению контроллерами внешних устройств и обмену данными между основной памятью и внешними устройствами. Устройства группируются по характерной скорости и подключаются к соответствующим каналам. «Быстрые» устройства (например, накопители на магнитных дисках) подсоединяются к селекторным каналам. Такое устройство получает селекторный канал в монопольное использование на все время выполнения операции обмена данными. «Медленные» устройства подключаются к мультиплексным каналам. *Мультиплексный канал* разделяется (мультиплексируется) между несколькими устройствам, при этом возможен одновременный обмен данными с несколькими устройствами. Доступ к оперативной памяти может получить и центральный процессор, и один из каналов. Для управления очередностью доступа имеется контроллер оперативной памяти. Он определяет приоритетную дисциплину доступа при одновременном обращении нескольких устройств к памяти. Наименьший приоритет имеет центральный процессор. Среди каналов больший приоритет имеют медленные каналы. Таким образом, приоритет обратно пропорционален частоте обращения устройств к памяти.

За счет существенного усложнения организации ЭВМ упрощается архитектура ввода-вывода. Связь между отдельными узлами осуществляется по схеме, напоминающей треугольник (см. рисунок 5.15).

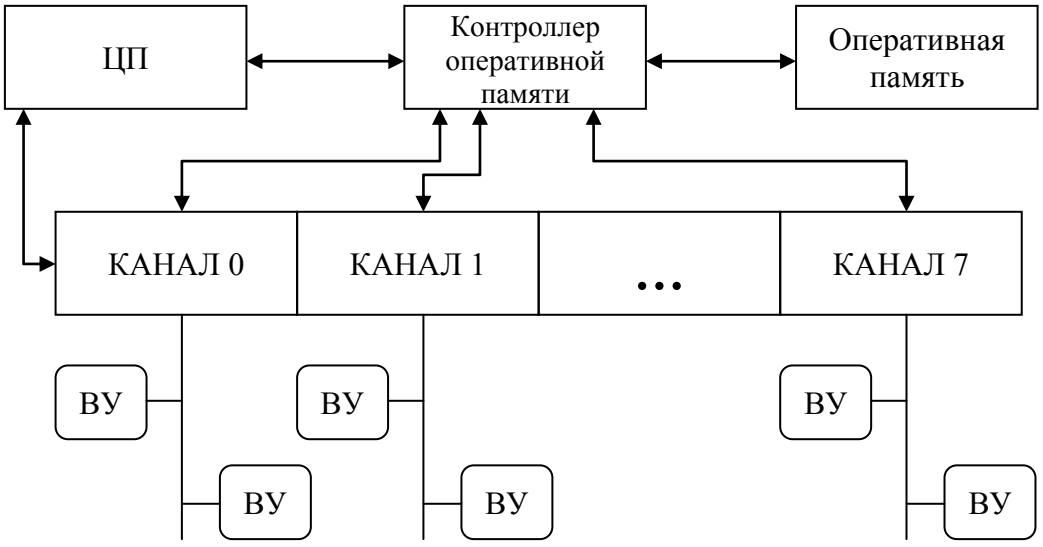


Рисунок 5.15 – Упрощенная схема ЭВМ с канальной организацией: ЦП – центральный процессор, ВУ – внешние устройства

Операции обмена данными становятся более простыми. Канал, по сути, представляет собой специализированный «интеллектуальный» контроллер прямого доступа к памяти (interrupt, INT, IRQ). Для ускорения обмена данными реализованы несколько трактов обмена данными (процессор – основная память и каналы - основная память). О своем состоянии канал может информировать процессор с помощью прерываний (Direct Memory

Access, DMA). Все контроллеры внешних устройств подключаются к «своим» каналам с помощью стандартного интерфейса. Свобода подключения внешних устройств сохраняется благодаря стандартному протоколу интерфейса, при этом появляется возможность группировать устройства по характеристикам.

Результатом введения каналов (специализированных процессоров ввода-вывода) является большая стандартизация и упрощение процессов обмена. С другой стороны, вводятся некоторые ограничения. Например, сохраняется только одна схема, напоминающая схему прямого доступа, с обменом информации между процессором и каналом по прерываниям.

Канал, являясь хотя и специализированным, но все-таки процессором, выполняет свою канальную программу. Она состоит из канальных команд и хранится в оперативной памяти. Длина канальной программы произвольна, последняя команда канальной программы содержит признак конца. Подготовку канальной программы и загрузку ее в оперативную память осуществляет операционная система. После того как канальная программа подготовлена, адрес ее начала размещается в фиксированной ячейке памяти, называемой словом адреса канала CAW (Channel Address Word).

Для управления каналами процессор имеет всего несколько команд. Операция обмена данными инициируется центральным процессором с помощью команды НАЧАТЬ ВВОД - ВЫВОД - SIO M,N (Start Input - Output). Операциями команды являются M – номер канала и N – номер устройства в канале. Выдав команду запуска обмена, процессор, не обращая внимания на обменный процесс, продолжает выполнять свою программу. Центральный процессор может проверить состояние канала с помощью команды ОПРОСИТЬ ВВОД - ВЫВОД - TIO (Test Input - Output).

Команда SIO M,N передается во все каналы, но воспринимает ее только канал M. Если канал занят, то он устанавливает соответствующее состояние своих регистров, и процессор по команде TIO может выяснить, что запуск канальной программы не состоялся. Если канал свободен, он выполняет следующие действия. Во-первых, выбирает из оперативной памяти CAW в свой регистр, во-вторых, передает подключенным к нему устройствам команду SIO. Команда запуска ввода-вывода SIO M,N передается всем устройствам, но воспринимает ее только устройство N. Если устройство занято или не готово, в регистрах канала устанавливается соответствующее состояние и процессор по команде TIO может узнать о том, что операция обмена данными не состоялась. Если же устройство свободно и готово к обмену данными, оно устанавливает в интерфейсе сигнал ожидания. Вся дальнейшая обменная операция протекает по инициативе внешнего устройства. Получив сигнал ожидания, канал выбирает по адресу CAW адрес канальной команды и передает ее в контроллер внешнего устройства, где она выполняется.

Канальные команды могут быть подготовительными или командами обмена данными. Подготовительные команды устанавливают режимы работы внешних устройств, осуществляют операции поиска и т.д.

Обменные команды содержат коды операций и адреса оперативной памяти. Обмен происходит по асинхронной схеме по инициативе внешнего устройства. Данные извлекаются из памяти и помещаются в нее напрямую, без посредников.

После выполнения команды канал проверяет в выполненной команде признак конца. Если это не последняя команда, меняется адрес CAW и выбирается следующая команда. Если команда последняя, канал «привлекает к себе внимание» процессора с помощью сигнала прерывания. По сигналу прерывания запускается обработчик, являющийся частью операционной системы. Обработчик прерывания выполняет операции, завершающие обмен.

Канал может сгенерировать сигнал прерывания до окончания канальной программы при возникновении исключительной ситуации. В этом случае операционная система запрашивает состояние регистров канала и выясняет, что именно произошло, и определяет, какие действия необходимо предпринять в возникшей ситуации.

Отметим некоторые особенности канальных машин. Несколько подряд идущих канальных команд могут образовывать цепочку данных. В этом случае имеется одна команда обмена, например чтения физической записи из нескольких адресов оперативной памяти со счетчиков. Одна физическая запись распределяется в несколько адресов оперативной памяти.

В ЭВМ с канальной организацией процессор практически полностью освобождается от рутинной работы по организации ввода-вывода. Управление контроллерами внешних устройств и обмен данными берет на себя канал. Наличие нескольких трактов передачи данных снимает трудности, связанные с блокировкой единственного тракта передачи данных (системной шины), что повышает скорость обмена. Все это дает возможность производить обмен данными с внешними устройствами параллельно с основной вычислительной работой центрального процессора. В результате общая производительность системы существенно возрастает. Удорожание схемы окупается.

Одной из первых машин с каналами была ЭВМ второго поколения IBM-704. Ярким примером ЭВМ с каналами являются машины семейства IBM-360/370. Появление этих ЭВМ произвело переворот в вычислительной технике, и на долгие годы они стали образцом для подражания у создателей ЭВМ. Хотя в настоящее время эти машины ушли в прошлое, они оставили богатое наследие в виде интересных архитектурных решений, программных и алгоритмических разработок. В настоящее время схемы со специализированными процессорами ввода-вывода часто встречаются в ЭВМ различных типов. Несомненно, идея схемы с каналами не умерла, и к ней еще неоднократно будут возвращаться.

5.4 Сетевые технологии обработки данных

На сегодняшний день в мире существует более 130 миллионов компьютеров и более 80% из них объединены в различные информационно-

вычислительные сети от малых локальных сетей в офисах до глобальных сетей типа Internet. Всемирная тенденция к объединению компьютеров в сети обусловлена рядом важных причин, таких как ускорение передачи информационных сообщений, возможность быстрого обмена информацией между пользователями, получение и передача сообщений (факсов, e-mail – электронных писем и пр.), не отходя от рабочего места, возможность мгновенного получения любой информации из любой точки земного шара, а также обмен информацией между компьютерами разных фирм производителей работающих под разным программным обеспечением.

Такие огромные потенциальные возможности, которые несет в себе вычислительная сеть и тот новый потенциальный подъем, который при этом испытывает информационный комплекс, а также значительное ускорение производственного процесса дают нам право принимать это к разработке и применять их на практике.

Поэтому необходимо разработать принципиальное решение вопроса по организации ИВС (информационно-вычислительной сети) на базе уже существующего компьютерного парка и программного комплекса, отвечающего современным научно-техническим требованиям с учетом возрастающих потребностей и возможностей дальнейшего постепенного развития сети в связи с появлением новых технических и программных решений.

Понятие локальной вычислительной сети

Под локальной вычислительной сетью (ЛВС) понимают совместное подключение нескольких отдельных компьютерных рабочих мест (рабочих станций) к единому каналу передачи данных. Благодаря вычислительным сетям мы получили возможность одновременного использования программ и баз данных несколькими пользователями.

Понятие *локальная вычислительная сеть* (англ. LAN - Local Area Network) относится к географически ограниченным (территориально или производственно) аппаратно-программным реализациям, в которых несколько компьютерных систем связаны друг с другом с помощью соответствующих средств коммуникаций. Благодаря такому соединению пользователь может взаимодействовать с другими рабочими станциями, подключенными к этой ЛВС.

В производственной практике ЛВС играют очень большую роль. Посредством ЛВС в систему объединяются персональные компьютеры, расположенные на многих удаленных рабочих местах, которые используют совместно оборудование, программные средства и информацию. Рабочие места сотрудников перестают быть изолированными и объединяются в единую систему. Рассмотрим преимущества, получаемые при сетевом объединении персональных компьютеров в виде внутрипроизводственной вычислительной сети.

Разделение ресурсов. Позволяет экономно использовать ресурсы,

например, управлять периферийными устройствами, такими как лазерные принтеры, со всех присоединенных рабочих станций.

Разделение данных. Разделение данных предоставляет возможность доступа и управления базами данных с периферийных рабочих мест, нуждающихся в информации.

Разделение программных средств. Разделение программных средств предоставляет возможность одновременного использования централизованных, ранее установленных программных средств.

Разделение ресурсов процессора. При разделении ресурсов процессора возможно использование вычислительных мощностей для обработки данных другими системами, входящими в сеть. Предоставляемая возможность заключается в том, что на имеющиеся ресурсы не «набрасываются» моментально, а только лишь через специальный процессор, доступный каждой рабочей станции.

Многопользовательский режим. Многопользовательские свойства системы содействуют одновременному использованию централизованных прикладных программных средств, ранее установленных и управляемых, например, если пользователь системы работает с другим заданием, то текущая выполняемая работа отодвигается на задний план.

Все ЛВС работают в одном стандарте, принятом для компьютерных сетей - в стандарте Open Systems Interconnection (OSI) - взаимодействия открытых систем.

Базовая модель OSI (Open System Interconnection)

Для того чтобы взаимодействовать, люди используют общий язык. Если они не могут разговаривать друг с другом непосредственно, они применяют соответствующие вспомогательные средства для передачи сообщений.

Показанные выше стадии необходимы, когда сообщение передается от отправителя к получателю.

Для того чтобы, привести в движение процесс передачи данных, использовали машины с одинаковым кодированием данных и связанные одна с другой. Для единого представления данных, в линиях связи по которым передается информация, сформирована Международная организация по стандартизации (англ. ISO - International Standards Organization).

Международная организация по стандартизации (ISO) разработала базовую (эталонную) модель взаимодействия открытых систем (англ. Open Systems Interconnection (OSI)). Эта модель является международным стандартом для передачи данных. Модель содержит семь отдельных уровней (таблица 5.1):

Таблица 5.1 – Модель OSI

Уровень 1	Физический	Битовые протоколы передачи информации
Уровень 2	Канальный	Формирование кадров, управление доступом к среде
Уровень 3	Сетевой	Маршрутизация, управление потоками данных
Уровень 4	Транспортный	Обеспечение взаимодействия удаленных процессов
Уровень 5	Сеансовый	Поддержка диалога между удаленными процессами
Уровень 6	Представления данных	Интерпретация передаваемых данных
Уровень 7	Прикладной	Пользовательское управление данными

Основная идея этой модели заключается в том, что каждому уровню отводится конкретная роль, в том числе и транспортной среде. Благодаря этому общая задача передачи данных расчленяется на отдельные, легко обозримые задачи. Необходимые соглашения для связи одного уровня, например, вышерасположенного и нижерасположенного, называют *протоколом*.

Так как пользователи нуждаются в эффективном управлении, система вычислительной сети представляется как комплексное строение, которое координирует взаимодействие задач пользователей.

С учетом вышеизложенного можно вывести следующую уровневую модель с административными функциями, выполняющимися в пользовательском прикладном уровне.

Отдельные уровни базовой модели проходят в направлении вниз от источника данных (от уровня 7 к уровню 1) и в направлении вверх от приемника данных (от уровня 1 к уровню 7). Пользовательские данные передаются в нижерасположенный уровень вместе со специфическим для уровня заголовком до тех пор, пока не будет достигнут последний уровень.

На приемной стороне поступающие данные анализируются и, по мере надобности, передаются далее в вышерасположенный уровень, пока информация не будет передана в пользовательский прикладной уровень.

Уровень 1. Физический. На физическом уровне определяются электрические, механические, функциональные и процедурные параметры для физической связи в системах. Физическая связь и неразрывная с ней эксплуатационная готовность являются основной функцией 1-го уровня. Стандарты физического уровня включают рекомендации V.24 МККТТ

(CCITT), EIA RS232 и X.21,а так же стандарт ISDN (Integrated Services Digital Network). В качестве среды передачи данных используют трехжильный медный провод (экранированная витая пара), коаксиальный кабель, оптоволоконный проводник и радиорелейную линию.

Уровень 2. Канальный. Канальный уровень формирует из данных, передаваемых 1-м уровнем, так называемые «кадры» и последовательности кадров. На этом уровне осуществляются управление доступом к передающей среде, используемой несколькими ЭВМ, синхронизация, обнаружение и исправление ошибок.

Уровень 3. Сетевой. Сетевой уровень устанавливает связь в вычислительной сети между двумя абонентами. Соединение происходит благодаря функциям маршрутизации, которые требуют наличия сетевого адреса в пакете. Сетевой уровень должен также обеспечивать обработку ошибок, мультиплексирование, управление потоками данных.

Протоколы сетевого уровня маршрутизируют данные от источника к получателю. Работающие на этом уровне устройства (маршрутизаторы) условно называют устройствами третьего уровня (по номеру уровня в модели OSI).

Протоколы сетевого уровня: IP/IPv4/IPv6 (Internet Protocol), IPX (Internetwork Packet Exchange, протокол межсетевого обмена), X.25 (частично этот протокол реализован на уровне 2), CLNP (сетевой протокол без организации соединений), IPsec (Internet Protocol Security). Протоколы маршрутизации - RIP (Routing Information Protocol), OSPF (Open Shortest Path First)

Рисунок 5.16 – Передача IP-пакетов на сетевом уровне не зависит от того, как соединены абоненты

Уровень 4. Транспортный. Транспортный уровень поддерживает непрерывную передачу данных между двумя взаимодействующими друг с другом пользовательскими процессами. Качество транспортировки, безошибочность передачи, независимость вычислительных сетей, сервис транспортировки из конца в конец, минимизация затрат и адресация связи гарантируют непрерывную и безошибочную передачу данных.

Протокол *UDP* (User Datagram Protocol) является одним из двух основных протоколов транспортного уровня, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, которые не многим отличаются от услуг, предоставляемых протоколом IP. Протокол UDP обеспечивает ненадежную доставку датаграмм и не поддерживает соединений из конца в конец. Другими словами, его пакеты могут быть потеряны, продублированы или прийти не в том порядке, в котором они были отправлены. К заголовку IP-пакета он добавляет два поля, одно из которых, поле "порт", обеспечивает мультиплексирование информации между разными прикладными процессами, а другое поле - "контрольная сумма" - позволяет поддерживать целостность данных. Примерами сетевых приложений, использующих UDP, являются NFS и SNMP.

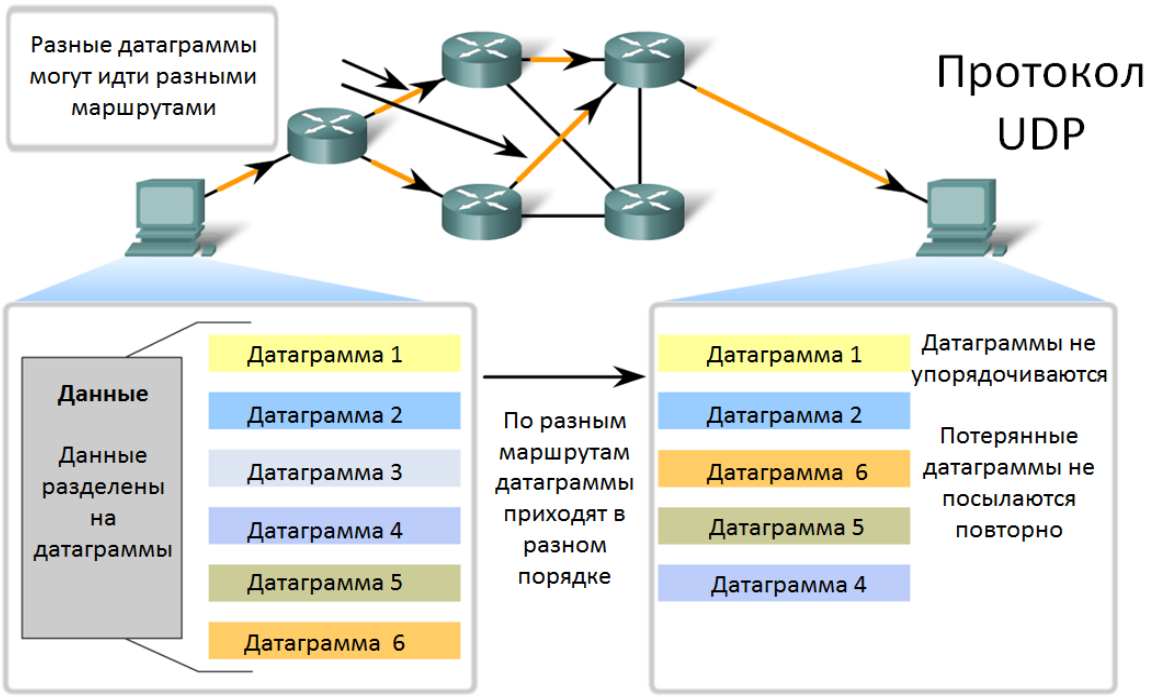


Рисунок 5.17 – Особенности протокола UDP

В стеке протоколов TCP/IP протокол *TCP* (Transmission Control Protocol) работает, как и протокол UDP, на транспортном уровне. Протокол TCP предоставляет транспортные услуги, отличающиеся от услуг UDP. Вместо ненадежной доставки датаграмм без установления соединений, он обеспечивает гарантированную доставку с установлением соединений между прикладными процессами в виде байтовых потоков.

Протокол TCP используется в тех случаях, когда требуется надежная доставка сообщений. Он освобождает прикладные процессы от необходимости использовать таймауты и повторные передачи для обеспечения надежности. Наиболее типичными прикладными процессами, использующими TCP, являются FTP и TELNET. Кроме того, TCP используют система X-Window, rcp (remote copy - удаленное копирование) и другие "r-

команды". Реализация TCP требует большой производительности процессора и большой пропускной способности сети. Внутренняя структура модуля TCP гораздо сложнее структуры модуля UDP.

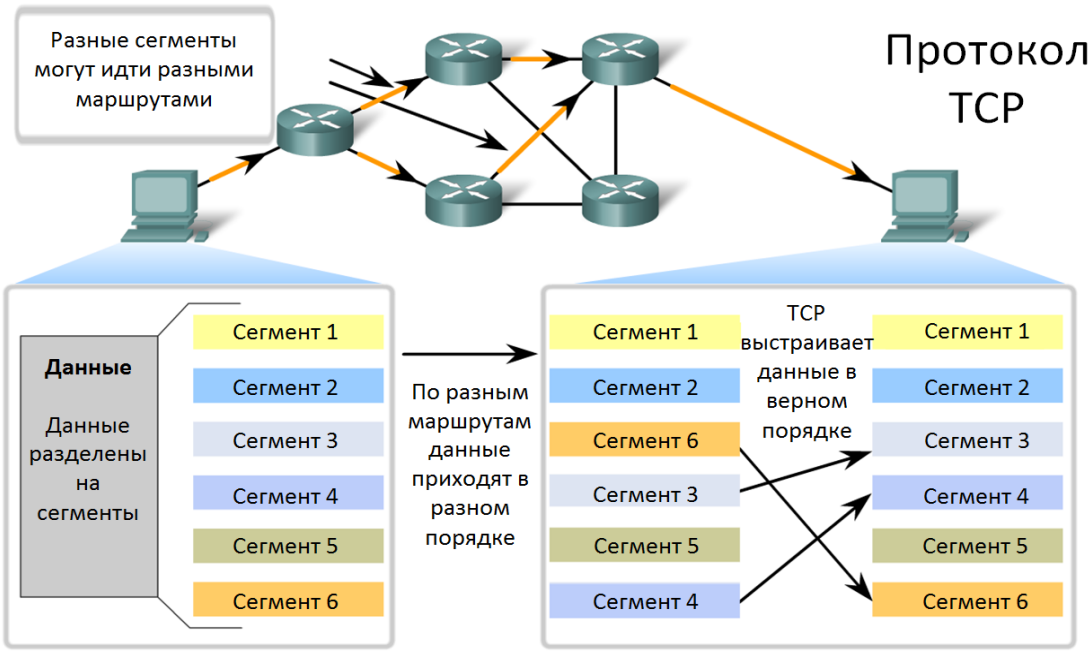


Рисунок 5.18 – Особенности протокола TCP

Уровень 5. Сеансовый. Сеансовый уровень координирует прием, передачу и выдачу одного сеанса связи. Для координации необходимы: контроль рабочих параметров, управление потоками данных промежуточных накопителей и диалоговый контроль, гарантирующий передачу имеющихся в распоряжении данных. Кроме того, сеансовый уровень содержит дополнительно функции управления паролями, подсчета платы за пользование ресурсами сети, управления диалогом, синхронизации и отмены связи в сеансе передачи после сбоя вследствие ошибок в нижерасположенных уровнях.

Уровень 6. Представления данных. Уровень представления данных предназначен для интерпретации данных; а также подготовки данных для пользовательского прикладного уровня. На этом уровне происходит преобразование данных из кадров, используемых для передачи данных в экранный формат или формат для печатающих устройств оконечной системы.

Уровень 7. Прикладной. В прикладном уровне необходимо предоставить в распоряжение пользователей уже переработанную информацию. С этим может справиться системное и пользовательское прикладное программное обеспечение.

Архитектура ЛВС

Архитектура сети описывает не только физическое расположение сетевых устройств, но и тип используемых адаптеров и кабелей. Кроме того, сетевая архитектура определяет методы передачи данных по кабелю.

Одноранговая сеть.

В одноранговой сети все компьютеры равноправны: нет иерархии среди компьютеров и нет выделенного сервера, и, как правило, каждый компьютер функционирует и как клиент и как сервер. Все пользователи самостоятельно решают, какие данные на своем компьютере сделать доступными для всех. Одноранговую сеть называют также рабочей группой. Рабочая группа - это небольшой коллектив, поэтому в одноранговой сети, как правило, не более 10 компьютеров.

Одноранговые сети относительно просты. Поскольку каждый компьютер является и клиентом и сервером, нет необходимости в мощном центральном сервере или в других компонентах, обязательных для более сложных сетей. Одноранговые сети обычно дешевле сетей на основе сервера, но требуют более мощных и дорогих компьютеров.

В одноранговой сети требования к производительности и к уровню защиты для сетевого программного обеспечения, как правило, ниже, чем в сетях с выделенным сервером. Выделенные серверы функционируют исключительно в качестве серверов, но не клиентов или рабочих станций.

Практически во все современные операционные системы встроена поддержка одноранговых сетей. Поэтому, чтобы установить одноранговую сеть дополнительного программного обеспечения не требуется.

Одноранговая компьютерная сеть в общем виде выглядит так:

- компьютеры расположены на рабочих столах пользователей;
- пользователи сами выступают в роли администраторов и сами обеспечивают защиту информации;
- для объединения компьютеров в сеть применяется простая кабельная система.

Если эти условия выполняются, то, скорее всего, выбор одноранговой сети будет правильным.

Защита подразумевает установку пароля на разделяемый ресурс, например, на каталог. Централизованно управлять защитой в одноранговой сети очень сложно, так как каждый пользователь устанавливает ее самостоятельно, да и общие ресурсы могут находиться на всех компьютерах, а не только на центральном сервере. Такая ситуация представляет серьезную угрозу для всей сети, кроме того некоторые пользователи могут вообще не устанавливать защиту.

Сети на основе сервера.

Если к сети подключено более 10 пользователей, то одноранговая сеть, где компьютеры выступают в роли клиентов, и серверов, может оказаться

недостаточно производительной. Поэтому большинство сетей используют выделенные серверы. *Выделенным* (dedicated) называется такой сервер, который функционирует только как сервер. Они специально оптимизированы для быстрой обработки запросов от сетевых клиентов и для управления защитой файлов и каталогов. Сети на основе сервера стали промышленным стандартом.

С увеличением размеров сети и объемов сетевого трафика необходимо увеличивать количество серверов. Распределение задач среди нескольких серверов гарантирует, что каждая задача будет выполняться самым эффективным способом из всех возможных.

Круг задач, которые должны выполнять серверы, многообразен и сложен. Чтобы приспособиться к возрастающим потребностям пользователей, серверы в больших сетях стали специализированными. Например, в сети на базе ОС Windows существуют различные типы серверов.

Файл-серверы и принт-серверы управляют доступом соответственно к файлам и принтерам, на серверах приложений выполняются прикладные части клиент-серверных приложений, а так же находятся данные, доступные клиентам. Например, чтобы упростить извлечение данных, серверы хранят большие объемы информации в структурированном виде. Эти серверы отличаются от файл-серверов и принт-серверов. В принт-серверах файл или данные целиком копируются на запрашиваемый компьютер. А в сервере приложений на запрашиваемый компьютер посылаются только результаты запроса. Приложение-клиент на удаленном компьютере получает доступ к данным, хранимым на сервере приложений. Однако вместо всей базы данных на ваш компьютер с сервера загружаются только результаты запроса.

В расширенной сети использование серверов различных типов становится наиболее актуальным. Поэтому необходимо учитывать всевозможные нюансы, которые могут проявиться при разрастании сети, с тем, чтобы изменение роли определенного сервера в дальнейшем не отразилось на работе всей сети.

Основным аргументом при работе в сети на основе выделенного сервера является, как правило, защита данных. В таких сетях проблемами безопасности может заниматься один администратор.

Поскольку жизненно важная информация расположена централизованно, то есть сосредоточена на одном или нескольких серверах, нетрудно обеспечить ее регулярное резервное копирование. Благодаря избыточным системам данные на любом сервере могут дублироваться в реальном времени, поэтому в случае повреждения основной области хранения данных информация не будет потеряна – легко воспользоваться резервной копией. Сети на основе сервера могут поддерживать тысячи пользователей. Сетью такого размера, будь она одноранговой, невозможно было бы управлять. Так как компьютер пользователя не выполняет функции сервера, требования к его характеристикам зависят от самого пользователя.

Топологии вычислительной сети

Топология типа «звезда»

Концепция топологии сети в виде звезды (star) пришла из области больших ЭВМ, в которой головная машина получает и обрабатывает все данные с периферийных устройств как активный узел обработки данных. Этот принцип применяется в системах передачи данных. Вся информация между двумя периферийными рабочими местами проходит через центральный узел вычислительной сети, который представляет собой коммутатор (ранее – концентратор).

Пропускная способность сети определяется вычислительной мощностью узла и гарантируется для каждой рабочей станции. Коллизий (столкновений) данных не возникает.

Кабельное соединение довольно простое, так как каждая рабочая станция связана с узлом. Затраты на прокладку кабелей могут быть довольно высокими, особенно когда центральный узел географически расположен не в центре топологии.

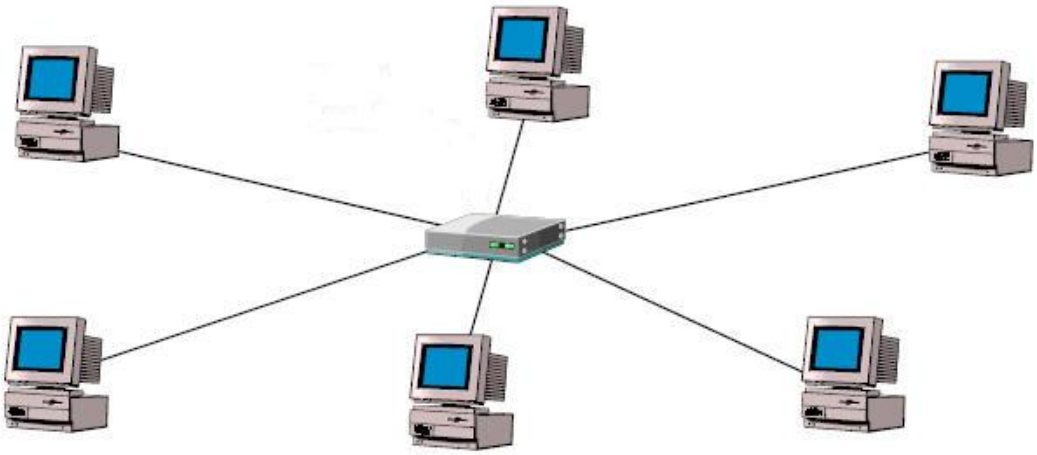


Рисунок 5.19 – Топология в виде звезды

При расширении вычислительных сетей не могут быть использованы ранее выполненные кабельные связи; к новому рабочему месту необходимо прокладывать отдельный кабель из центра сети.

Топология в виде звезды является наиболее популярной и самой быстродействующей из всех топологий вычислительных сетей, поскольку передача данных между рабочими станциями проходит через центральный узел (при его хорошей производительности) по отдельным линиям, используемым только этими рабочими станциями. Частота запросов передачи информации от одной станции к другой невысокая по сравнению с достигаемой в других топологиях.

Производительность вычислительной сети в первую очередь зависит от мощности коммутатора. Он может быть узким местом вычислительной сети. В случае выхода из строя центрального узла нарушается работа всей сети. Вместе с тем при обрыве связи с одной из станций вся остальная сеть будет

работать без малейших проблем.

Кольцевая топология

При кольцевой (ring) топологии сети рабочие станции связаны одна с другой по кругу, т.е. рабочая станция 1 с рабочей станцией 2, рабочая станция 3 с рабочей станцией 4 и т.д. Последняя рабочая станция связана с первой. Коммуникационная связь замыкается в кольцо.

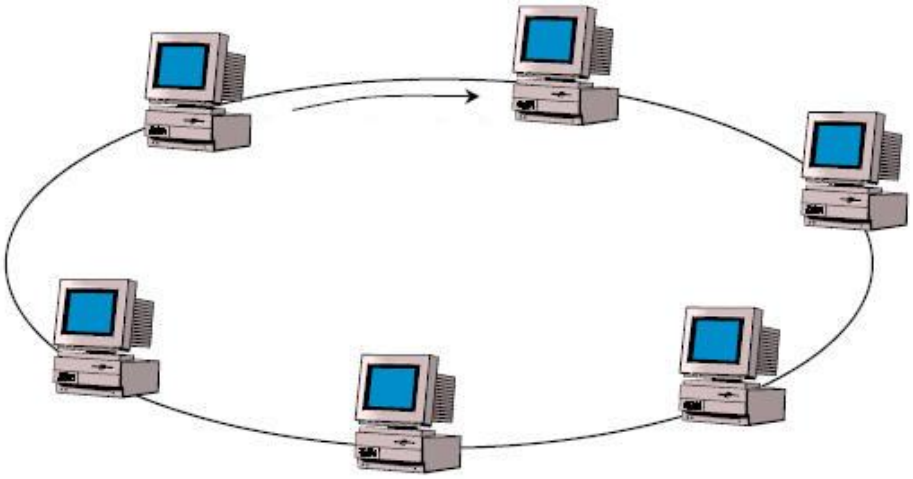


Рисунок 5.20 – Кольцевая топология

Прокладка кабелей от одной рабочей станции до другой может быть довольно сложной и дорогостоящей, особенно если географически рабочие станции расположены далеко от кольца (например, в линию).

Сообщения циркулируют регулярно по кругу. Рабочая станция посылает по определенному конечному адресу информацию, предварительно получив из кольца запрос. Пересылка сообщений является очень эффективной, так как большинство сообщений можно отправлять «в дорогу» по кабельной системе одно за другим. Очень просто можно сделать кольцевой запрос на все станции. Продолжительность передачи информации увеличивается пропорционально количеству рабочих станций, входящих в вычислительную сеть.

Основная проблема при кольцевой топологии заключается в том, что каждая рабочая станция должна активно участвовать в пересылке информации, и в случае выхода из строя хотя бы одной из них вся сеть парализуется. Неисправности в кабельных соединениях локализуются легко. Часто используют двойное двунаправленное кольцо для того, чтобы проблема со станцией или обрыв провода не приводил к параличу всей сети.

Подключение новой рабочей станции требует краткосрочного выключения сети, так как во время установки кольцо должно быть разомкнуто. Ограничения на протяженность вычислительной сети не существует, так как оно, в конечном счете, определяется исключительно расстоянием между двумя рабочими станциями.

Шинная топология

При шинной (bus) топологии среда передачи информации представляется в форме коммуникационного пути, доступного для всех рабочих станций, к которому они все должны быть подключены. Все рабочие станции могут непосредственно вступать в контакт с любой рабочей станцией, имеющейся в сети.

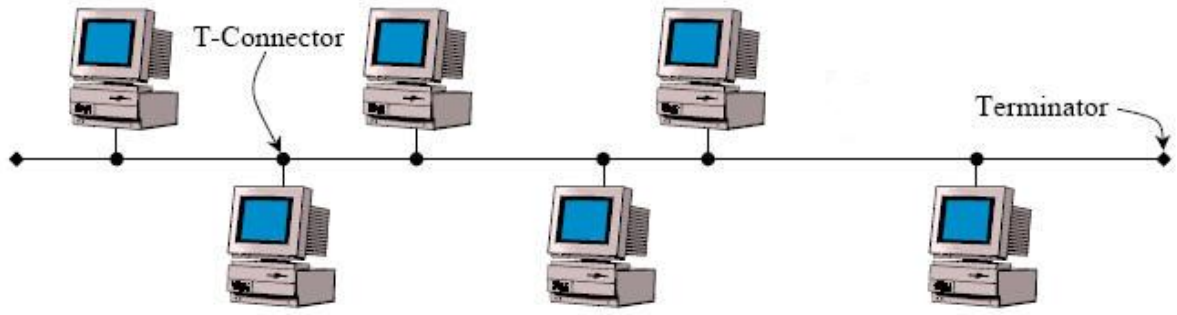


Рисунок 5.21 – Шинная топология

Рабочие станции в любое время, без прерывания работы всей вычислительной сети могут быть подключены к ней или отключены. Функционирование вычислительной сети не зависит от состояния отдельной рабочей станции. К окончанию магистрального провода с двух сторон подключены специальные устройства - терминаторы, представляющие собой резисторы сопротивлением 50 Ом для согласования нагрузки и предотвращения отражения сигнала.

В стандартной ситуации для шинной сети Ethernet часто используют тонкий кабель или Cheapernet-кабель (от англ. Cheap - дешевый) с тройниковым соединителем (или Т-соединением). Выключение и особенно подключение к такой сети требуют разрыва шины, что вызывает нарушение циркулирующего потока информации и зависание системы. Любой обрыв шины также приводит к остановке обмена информацией в сети.

В настоящее время топология "шина" практически не встречается.

Смешанная структура ЛВС

Наряду с известными топологиями вычислительных сетей - кольцо, звезда и шина, на практике применяется и комбинированная, например, гибридная структура. Она образуется в основном в виде комбинаций вышеназванных топологий вычислительных сетей. Вычислительные сети с гибридной структурой применяются там, где нерационально (или невозможно) непосредственное применение базовых сетевых структур в чистом виде.

Ячеистая (mesh) топология сети напоминает структуру соединения узлов в Internet, т.е. между узлами существует два и более маршрутов прохождения пакетов.

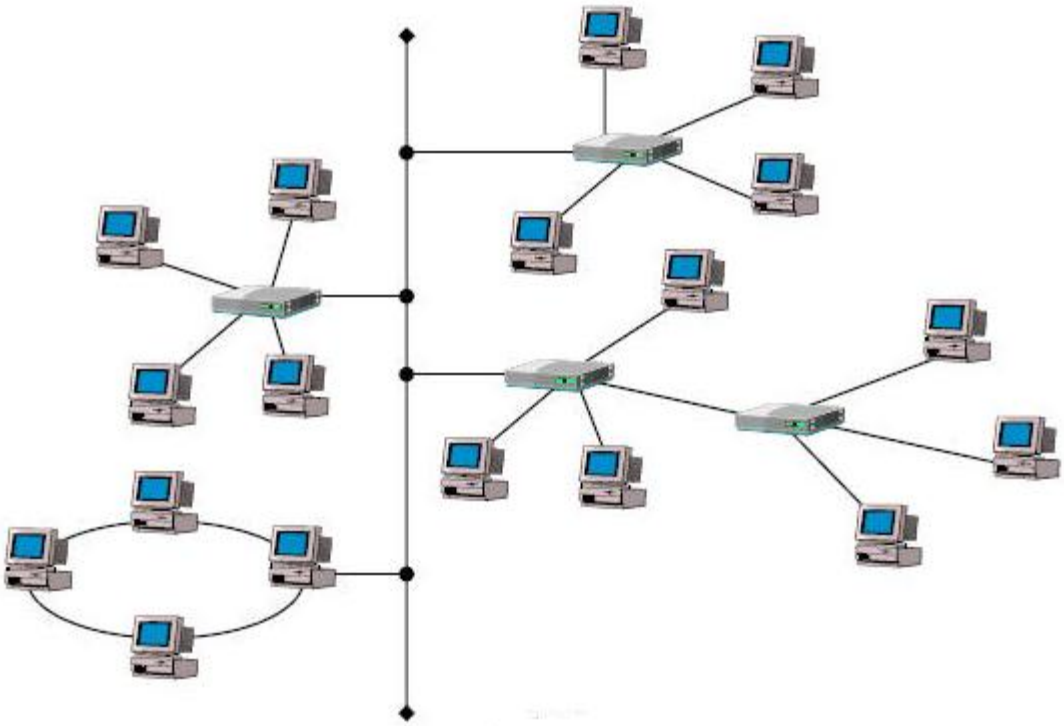


Рисунок 5.22 – Гибридная топология

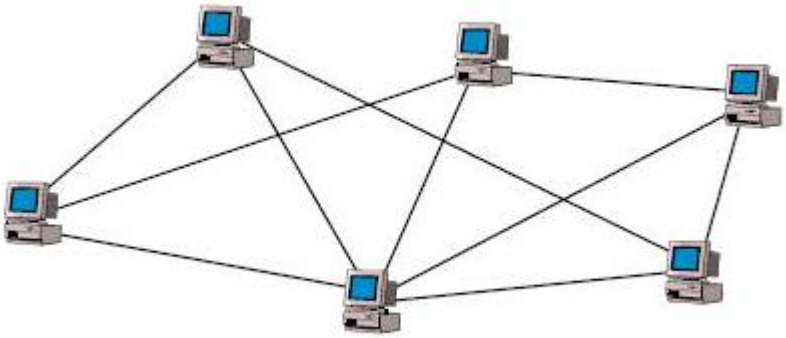


Рисунок 5.23 – Ячеистая топология

Сетевые устройства и средства коммуникаций

В качестве средств коммуникации наиболее часто используются витая пара, коаксиальный кабель, оптоволоконные линии. При выборе типа кабеля учитывают следующие показатели:

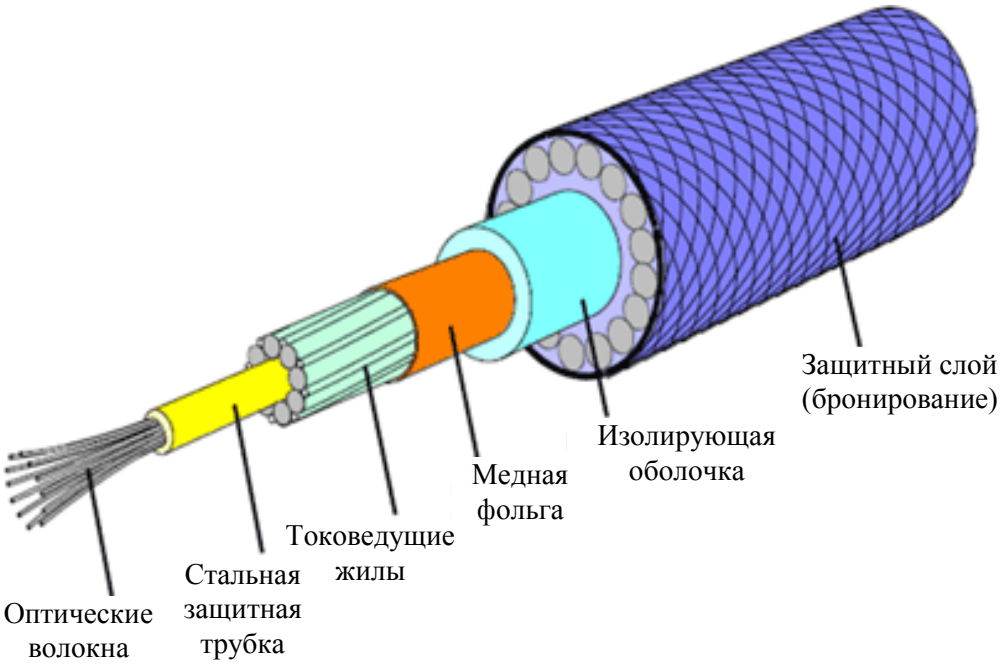
- стоимость монтажа и обслуживания;
- скорость передачи информации;
- ограничения на величину расстояния передачи информации без дополнительных усилителей-повторителей (репитеров/repeater);
- безопасность передачи данных.

Главная проблема заключается в одновременном обеспечении этих показателей, например, наивысшая скорость передачи данных ограничена максимально возможным расстоянием передачи данных, при котором еще обеспечивается требуемый уровень защиты данных. Легкая наращиваемость

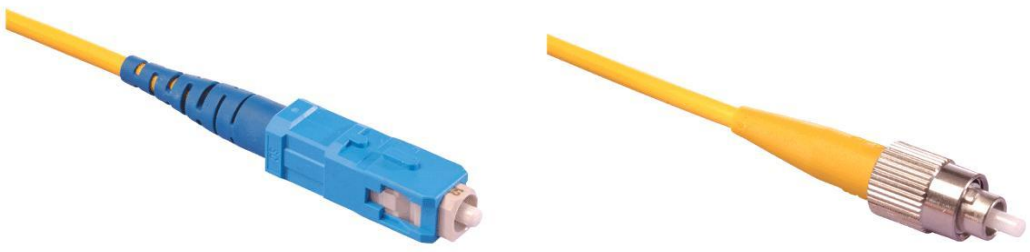
и простота расширения кабельной системы влияют на ее стоимость.

Виды используемых кабелей и сетевого оборудования

Оптоволоконные линии. Наиболее дорогими являются оптопроводники, называемые также стекловолоконным кабелем. Скорость распространения информации по ним достигает десятков гигабит в секунду. Допустимое удаление более 50 км. Внешнее воздействие помех практически отсутствует. На данный момент это наиболее дорогостоящее соединение для ЛВС. Применяются они там, где возникают электромагнитные поля помех или требуется передача информации на очень большие расстояния без использования повторителей. Они обладают противоподслушивающими свойствами, так как техника ответвлений в оптоволоконных кабелях очень сложна. Оптопроводники объединяются в ЛВС с помощью звездообразного соединения.



а)



б)

Рисунок 5.24 – а) оптоволоконный кабель для внешней прокладки;
б) SC- и FC-коннекторы

Витая пара. Наиболее дешевым и самым популярным в настоящее время кабельным соединением является витое двухжильное проводное соединение часто называемое «витой парой» (twisted pair или unshielded twisted pair/UTP). Она позволяет передавать информацию со скоростью до 1000 Мбит/с, легко наращивается, однако не защищена от помех. Длина кабеля не может превышать 1000 м при скорости передачи 1 Мбит/с (ограничение для ЛВС обычно составляет 100 м). Преимуществами являются низкая цена и простота установки. Для повышения помехозащищенности информации часто используют экранированную витую пару (shielded twisted pair/STP), т.е. витую пару, помещенную в экранирующую оболочку, подобно экрану коаксиального кабеля. Это увеличивает стоимость витой пары, однако позволяет достичь скоростей в десятки Гигабит/с (рисунок 5.25). Разъемы 8P8C (обычно называемые RJ-45) очень компактны, имеют пластмассовый корпус и восемь миниатюрных контактных площадок.

Рисунок 5.25 – Неэкранированная и экранированная витая пара

Ethernet-кабель. Ethernet-кабель также является коаксиальным кабелем с волновым сопротивлением 50 Ом. Его называют еще толстый Ethernet (thick), желтый кабель (yellow cable) или 10Base-T5 . Он использует 15-контактное стандартное включение. Вследствие помехозащищенности он является дорогой альтернативой обычным коаксиальным кабелям. Максимально доступное расстояние без повторителя не превышает 500 м, а общее расстояние сети Ethernet - около 3000 м. Ethernet-кабель благодаря своей магистральной топологии использует в конце лишь один нагрузочный резистор (терминатор).

Для подключения компьютера к толстому кабелю используется дополнительное устройство, называемое *трансивером*. Трансивер подсоединен непосредственно к сетевому кабелю. От него к компьютеру идет специальный трансиверный кабель, максимальная длина которого 50 м. На обоих его концах находятся 15-контактные DIX-разъемы (Digital, Intel и Xerox). С помощью одного разъема осуществляется подключение к трансиверу, с помощью другого - к сетевой плате компьютера.

Трансиверы освобождают от необходимости подводить кабель к каждому компьютеру. Расстояние от компьютера до сетевого кабеля

определяется длиной трансиверного кабеля.

Создание сети при помощи трансивера очень удобно. Он может в любом месте в буквальном смысле «пропускать» кабель. Эта простая процедура занимает мало времени, а получаемое соединение оказывается очень надежным.

Кабель не режется на куски, его можно прокладывать, не заботясь о точном месторасположении компьютеров, а затем устанавливать трансиверы в нужных местах. Крепятся трансиверы, как правило, на стенах, что предусмотрено их конструкцией.

Cheapernet-кабель. Более дешевым, чем Ethernet-кабель, является соединение Cheapernet-кабель или, как его часто называют, тонкий (thin) Ethernet, или 10Base-T2. Это также 50-омный коаксиальный кабель со скоростью передачи информации 10 Мбит/с (рисунок 5.26).

Тонкий Ethernet использует кабель типа RG-58A/V (диаметром 0,2 дюйма). Для маленькой сети используется кабель с сопротивлением 50 Ом. Коаксиальный кабель прокладывается от компьютера к компьютеру. У каждого компьютера оставляют небольшой запас кабеля на случай возможности его перемещения. Длина сегмента 185 м, количество компьютеров, подключенных к шине, - до 30.

После присоединения всех отрезков кабеля с BNC-коннекторами (Bayonel-Neill-Concelnan) к Т-коннекторам (название обусловлено формой разъема, похожей на букву «Т») получится единый кабельный сегмент. На его обоих концах устанавливаются терминаторы. Терминатор конструктивно представляет собой BNC-коннектор (он также надевается на Т-коннектор) со впаянным сопротивлением. Значение этого сопротивления должно соответствовать значению волнового сопротивления кабеля, т.е. для Ethernet нужны терминаторы с сопротивлением 50 Ом.

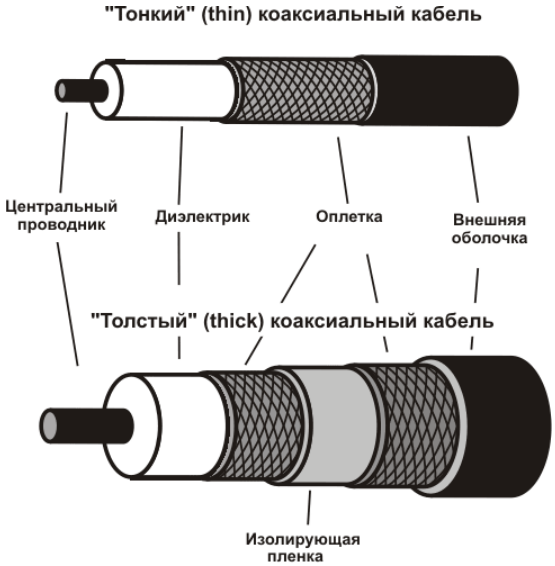
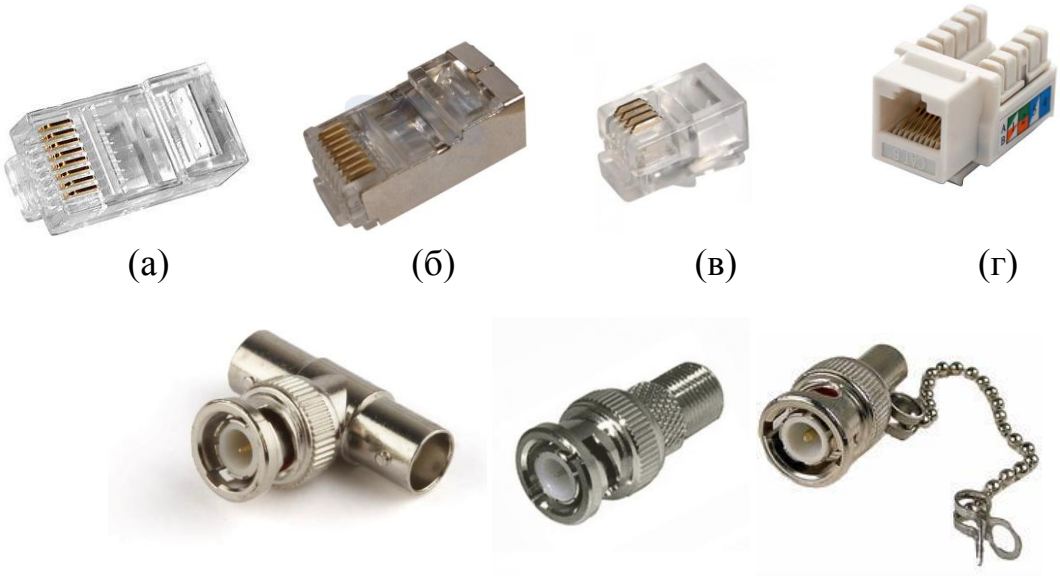


Рисунок 5.26 – Коаксиальный кабель

Расстояние между двумя рабочими станциями без повторителей может

составлять максимум 300 м (для ЛВС ограничение составляет 185 м), а общее расстояние для сети на Cheapernet-кабеля – около 1000 м. Приемопередатчик Cheapernet расположен на сетевой плате как для гальванической развязки между адаптерами, так и для усиления внешнего сигнала.

Разъемы. Электрический соединитель (иногда его называют "коннектор" от англ. connector) — электромеханическое устройство для осуществления соединения электрических проводников сетевых кабелей. Обычно состоит из вилки (штекера) и соответствующей ей розетки (гнезда).



(а) (б) (в) (г)

(д) (е) (ж)

Рисунок 5.27 – Разъемы для витой пары и коаксиального кабеля
а – коннектор для витой пары, *б* – коннектор для экранированной витой пары, *в* – телефонный коннектор, *г* – розеточный модуль для витой пары, *д* – Т-коннектор для коаксиального кабеля, *е* – BNC-коннектор для коаксиального кабеля, *ж* – терминатор для коаксиального кабеля

Сетевая карта. Платы сетевого адаптера выступают в качестве физического интерфейса или соединения между компьютером и сетевым кабелем. Платы вставляются в специальные гнезда (слоты расширения) всех компьютеров и серверов. Чтобы обеспечить физическое соединение между компьютером и сетью, к соответствующему разъему, или порту, платы (после ее установки) подключают сетевой кабель. Назначение платы сетевого адаптера:

- подготовка данных, поступающих от компьютера, к передаче по сетевому кабелю;
- передача данных другому компьютеру;
- управление потоком данных между компьютером и кабельной системой;
- плата сетевого адаптера принимает данные из сетевого кабеля и переводит в форму, понятную центральному процессору компьютера.

(а) (б)

Рисунок 5.28 – Плата сетевого адаптера
а - для проводного подключения, *б* – для беспроводного подключения

Плата сетевого адаптера состоит из аппаратной части и встроенных программ, записанных в ПЗУ (постоянном запоминающем устройстве). Эти программы реализуют функции подуровней управления логической связью и управление доступом к среде канального уровня модели OSI. В современных материнских платах сетевой адаптер встроенный.

Каждый сетевой адаптер имеет уникальный MAC-адрес. MAC-адрес (англ. Media Access Control — управление доступом к среде) – это уникальный идентификатор, присваиваемый каждой единице активного оборудования компьютерных сетей (его имеет любое устройство, которое умеет работать в сети). В широковещательных сетях (таких, как сети на основе Ethernet) MAC-адрес позволяет уникально идентифицировать каждый узел сети и доставлять данные только этому узлу. Таким образом, MAC-адреса формируют основу сетей на канальном уровне,

Существует распространенное мнение, что MAC-адрес жестко "вшит" в сетевой адаптер и сменить его нельзя или можно только с помощью программатора. На самом деле это не так. MAC-адрес легко меняется программным путем, так как значение, указанное через драйвер, имеет более высокий приоритет, чем присвоенное адаптеру производителем. Однако всё же существует оборудование, в котором смену MAC-адреса произвести невозможно иначе, как воспользовавшись программатором. Обычно это телекоммуникационное оборудование, например, приставки для IP-TV.

Разветвитель (концентратор, хаб, HAB). Он служит классическим центральным узлом в сетях с топологией «звезда». В настоящее время хабы полностью вытеснены коммутаторами.

Сетевой коммутатор (**свитч** от *switch* – переключатель) – устройство, предназначенное для соединения нескольких узлов компьютерной сети (обычно для топологии "звезда") в пределах одного или нескольких сегментов сети.

Рисунок 5.29 – Сетевые коммутаторы

В отличие от концентратора, который распространяет трафик от одного подключенного устройства ко всем остальным (что является абсолютно небезопасным с точки зрения защиты информации) , коммутатор передаёт данные только непосредственно получателю, исключение составляет широковещательный трафик (на MAC-адрес FF:FF:FF:FF:FF:FF) всем узлам сети. Это повышает производительность и безопасность сети, избавляя остальные сегменты сети от необходимости (и возможности) обрабатывать данные, которые им не предназначались.

Коммутатор – центральное устройство в сети на витой паре, от которого зависит ее работоспособность. Располагать его надо в легкодоступном месте, чтобы можно было легко подключать кабель и следить за индикацией портов. Коммутаторы выпускаются на разное количество портов – 5, 8, 12, 16, 24 и 48, соответственно к нему можно подключить такое же количество компьютеров.

Коммутаторы можно соединять друг с другом в каскадную схему, увеличивая количество подключаемых к сети компьютеров.

Репитер. При передаче по сетевому кабелю электрический сигнал постепенно ослабевает (затухает) и искажается до такой степени, что компьютер перестает его воспринимать. При необходимости охватить локальной сетью площадь большую, чем это позволяют рассматриваемые кабельные системы, применяется дополнительные устройства - *репитеры*, *повторители* (от англ. Repeat – повторять). Репитер имеет 2-портовое исполнение, т.е. он может объединить 2 сегмента по 185 м. Сегмент подключается к репитеру через Т-коннектор. К одному концу Т-коннектора подключается сегмент, а на другом ставится терминатор.

В сети может быть не больше четырех репитеров. Это позволяет получить сеть максимальной протяженностью 925 м.

Репитеры очень полезны, но злоупотреблять ими не стоит, так как они приводят к замедлению работы в сети.

Типы построения сетей по методам передачи информации

Локальная сеть Token Ring

Этот стандарт разработан фирмой IBM. В качестве передающей среды применяется неэкранированная или экранированная витая пара (UTP или STP), или оптоволокно. Скорость передачи данных 4 Мбит/с или 16 Мбит/с. В настоящее время считается существенно устаревшей технологией.

В качестве метода управления доступом станций к передающей среде используется метод «маркерное кольцо» (Token Ring). Основные положения этого метода:

- устройства подключаются к сети по топологии кольцо;
- все устройства, подключенные к сети, могут передавать данные, только получив разрешение на передачу (маркер);
- в любой момент времени только одна станция в сети обладает таким правом.

В IBM Token Ring используются три основных типа пакетов:

- пакет управление/данные (Data/Command Frame) – с помощью такого пакета выполняется передача данных или команд управления работой сети;
- маркер (Token) – станция может начать передачу данных только после получения такого пакета. В одном кольце может быть только один маркер и, соответственно, только одна станция с правом передачи данных;
- пакет сброса (Abort) – Посылка такого пакета называет прекращение любых передач.

Локальная сеть Ethernet

Ethernet – изначально коллизионная технология, основанная на общей шине, к которой компьютеры подключаются и «борются» между собой за право передачи пакета. Основной протокол - CSMA/CD (Carrier Sense Multiple Access with Collision Detection – множественный доступ с чувствительностью несущей и обнаружением коллизий). Дело в том, что если две станции одновременно начнут передачу, то возникает ситуация **коллизии**, и сеть некоторое время «ждет», пока «улягутся» переходные процессы и опять наступит «тишина». Существует еще один метод доступа - CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) - то же, но с исключением коллизий. Этот метод применяется в беспроводной технологии Radio Ethernet или Apple Local Talk – перед отправкой любого пакета в сети пробегает анонс о том, что сейчас будет происходить передача, и станции уже не пытаются ее инициировать.

Ethernet бывает полудуплексный (Half Duplex) по всем средам передачи; источник и приемник «говорит по очереди» (классическая коллизионная технология) и полнодуплексный (Full Duplex), когда две пары приемника и передатчика на устройствах говорят одновременно. Этот механизм работает только на витой паре (одна пара на передачу, одна пара на

прием) и на оптоволокне (одна пара на передачу, одна пара на прием).

Ethernet различается по скоростям и методам кодирования для различной физической среды, а также по типу пакетов (Ethernet II, 802.3, RAW, 802.2 (LLC), SNAP).

Ethernet различается по скоростям: 10 Мбит/с, 100 Мбит/с, 1000 Мбит/с (1 Гбит/с), 10Гбит/с и т.д. Для любой сети Ethernet могут быть использованы витая пара, одномодовое (SMF) или многомодовое (MMF) оптоволокно. В зависимости от этого существуют различные спецификации:

- 10 Мбит/с Ethernet: 10Base-T, 10Base-FL (10Base-2 и 10Base-5 существуют для коаксиального кабеля и уже не применяются);
- 100 Мбит/с Ethernet: 100Base-TX, 100Base-FX, 100Base-T4, 100Base-T2;
- Gigabit Ethernet: 1000Base-LX, 1000Base-SX (по оптике) и 1000Base-TX (для витой пары).

Технология Fast Ethernet IEEE 802.3U. Технология Fast Ethernet была стандартизирована комитетом IEEE 802.3. Стандарт получил название IEEE 802.3U. В сети Fast Ethernet организуются несколько доменов конфликтов (коллизий), но с обязательным учетом класса повторителя, используемого в доменах.

Репитеры Fast Ethernet (IEEE 802.3U) бывают двух классов и различаются по задержке в микросекундах. Соответственно в сегменте (логическом) может быть до двух репитеров класса 2 и один репитер класса 1. Для Ethernet (IEEE 802.3) сеть подчиняется правилу 5-4-3-2-1.

Правило 5-4-3-2-1 гласит: между любыми двумя рабочими станциями не должно быть более 5 физических сегментов, 4 репитеров (концентраторов), 3 «населенных» физических сегментов, 2 «населенных» межрепитерных связей (IRL), и все это должно представлять собой один коллизионный домен. Физически из концентратора выходит много проводов, но логически это все один сегмент Ethernet и один коллизионный домен, в связи с ним любой сбой одной станции отражается на работе других. Поскольку все станции вынуждены «слушать» чужие пакеты, коллизия происходит в пределах всего концентратора (на самом деле на другие порты посылается сигнал "Jam" (коллизия), но это не меняет сути дела). Сеть из 20 компьютеров, собранная на репитерах 100 Мбит/с, может работать медленнее, чем сеть из 20 компьютеров, включенных в коммутатор 10 Мбит/с. Если раньше считалось «нормальным» присутствие в сегменте до 30 компьютеров, то в нынешних сетях даже три рабочие станции могут загрузить весь сегмент.

Технология Gigabit Ethernet. Следующий шаг в развитии технологии Ethernet - стандарт IEEE-802.32. Данный стандарт предусматривает скорость обмена информацией между станциями локальной сети 1Гбит/с. Устройства Gigabit Ethernet объединяют сегменты сетей с Fast Ethernet со скоростями 100 Мбит/с. Используются сетевые карты со скоростью 1 Гбит/с, а также серия

сетевых устройств, таких как коммутаторы и маршрутизаторы. В сети с Gigabit Ethernet используется управление трафиком, контроль перегрузок и обеспечение качества обслуживания (Quality Of Service- QOS).

10-гигабитный Ethernet (Ethernet 10G, 10 Гбит/с)

Новый стандарт 10-гигабитного Ethernet включает в себя семь стандартов физической среды для LAN, MAN и WAN. В настоящее время он описывается поправкой IEEE 802.3ae и должен войти в следующую ревизию стандарта IEEE 802.3.

10GBASE-CX4 — технология 10-гигабитного Ethernet для коротких расстояний (до 15 метров), используется медный кабель CX4 и коннекторы InfiniBand.

10GBASE-SR — технология 10-гигабитного Ethernet для коротких расстояний (до 26 или 82 метров, в зависимости от типа кабеля), используется многомодовое волокно. Он также поддерживает расстояния до 300 метров с использованием нового многомодового волокна (2000 МГц/км).

10GBASE-LX4 — использует уплотнение по длине волны для поддержки расстояний от 240 до 300 метров по многомодовому волокну. Также поддерживает расстояния до 10 километров при использовании одномодового волокна.

10GBASE-LR и 10GBASE-ER — эти стандарты поддерживают расстояния до 10 и 40 километров соответственно.

10GBASE-SW, 10GBASE-LW и 10GBASE-EW — эти стандарты используют физический интерфейс, совместимый по скорости и формату данных с интерфейсом OC-192 / STM-64 SONET/SDH. Они подобны стандартам 10GBASE-SR, 10GBASE-LR и 10GBASE-ER соответственно, так как используют те же самые типы кабелей и расстояния передачи.

10GBASE-T, IEEE 802.3an-2006 — принят в июне 2006 года после 4 лет разработки. Использует витую пару категории 6 (максимальное расстояние 55 метров) и 6a (максимальное расстояние 100 метров).

10GBASE-KR — технология 10-гигабитного Ethernet для кросс-плат (backplane/midplane) модульных коммутаторов/маршрутизаторов и серверов (Modular/Blade).

Каковы перспективы развития Ethernet? Согласно оценкам экспертов, требования к полосе пропускания для вычислительных задач и приложений ядра сети растут с разными скоростями, что определяет необходимость двух соответствующих стандартов для следующих поколений Ethernet — 40 Gigabit Ethernet (или 40GbE) и 100 Gigabit Ethernet (или 100GbE). В настоящее время серверы, высокопроизводительные вычислительные кластеры, сетевые хранилища используют технологии 1GbE и 10GbE. Для существенного повышения скоростей придется решить немало проблем в области физики распространения сигналов.

5.5 IPv4-адресация

Для обмена данными в Интернете узлу необходим IP-адрес. Это логический сетевой адрес конкретного узла. Для обмена данными с другими устройствами, подключенными к Интернету, необходим правильно настроенный, уникальный IP-адрес.

IP-адрес присваивается сетевому интерфейсу узла. Обычно это сетевая плата, установленная в устройстве. Примерами пользовательских устройств с сетевыми интерфейсами могут служить рабочие станции, серверы, сетевые принтеры и IP-телефоны. Иногда в серверах устанавливают несколько сетевых плат, у каждой из которых есть свой IP-адрес. У интерфейсов маршрутизатора, обеспечивающего связь с сетью IP, также есть IP-адрес.

В каждом отправленном по сети пакете есть IP-адрес источника и назначения. Эта информация необходима сетевым устройствам для передачи информации по назначению и передачи источнику ответа

IP-адрес представляет собой серию из 32 двоичных бит (единиц и нулей). Человеку прочесть двоичный IP-адрес очень сложно. Поэтому 32 бита группируются по четыре 8-битных байта, в так называемые октеты. Читать, записывать и запоминать IP-адреса в таком формате людям сложно. Чтобы облегчить понимание, каждый октет IP-адреса представлен в виде своего десятичного значения. Октеты разделяются десятичной точкой или запятой. Это называется точечно-десятичной нотацией.

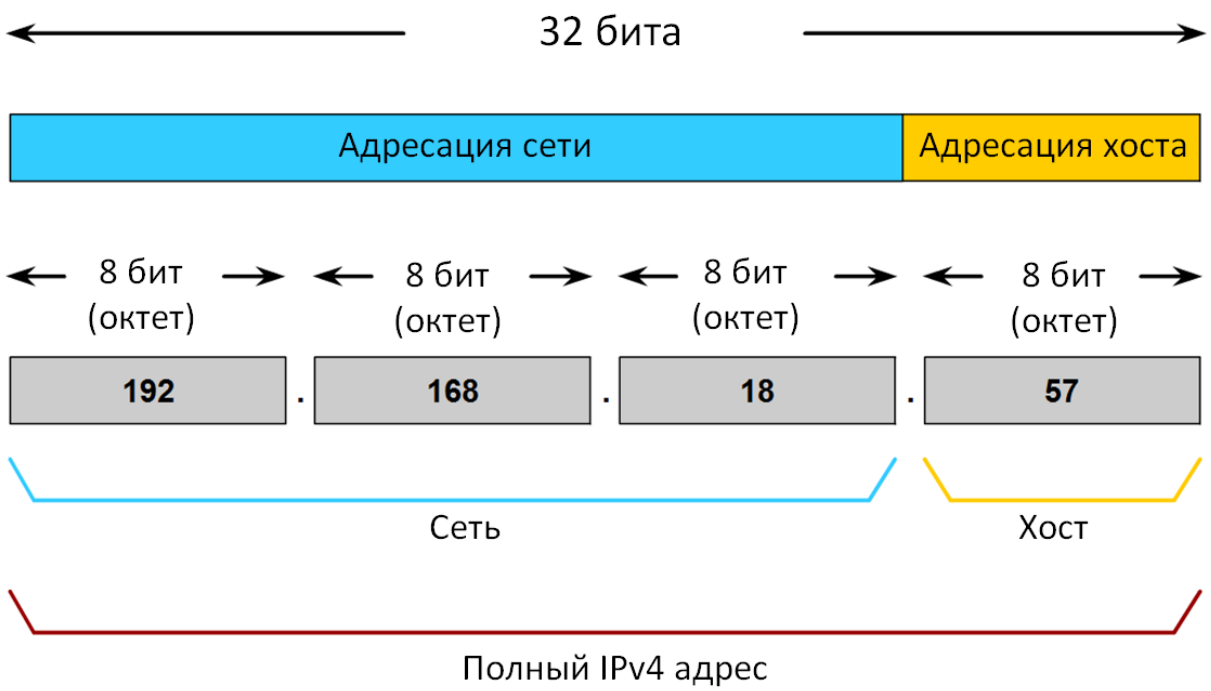


Рисунок 5.30 – Схема IPv4-адресации

При настройке IP-адрес узла вводится в виде десятичного числа с точками, например, 192.168.1.5. Представим, что нам пришлось бы вводить 32-битный двоичный эквивалент адреса -

11000000101010000000000100000101. Если ошибиться хотя бы в одном бите, получится другой адрес, и узел, возможно, не сможет работать в сети.

Структура 32-битного IP-адреса определяется межсетевым протоколом 4-ой версии (IPv4). На данный момент это один из самых распространенных в Интернете типов IP-адресов. По 32-битной схеме адресации можно создать более 4 миллиардов IP-адресов

Получая IP-адрес, узел просматривает все 32 бита по мере поступления на сетевой адаптер. Напротив, людям приходится преобразовывать эти 32 бита в десятичные эквиваленты, то есть в четыре октета. Каждый октет состоит из 8 бит, каждый бит имеет значение. У четырех групп из 8 бит есть один и тот же набор значений. Значение крайнего правого бита в октете – 1, значения остальных, слева направо – 2, 4, 8, 16, 32, 64 и 128.

Чтобы определить значение октета, нужно сложить значения позиций, где присутствует двоичная единица. Это соответствует $11111111_2 = 2^8 = 256$. Таким образом, значение каждого из четырех октетов находится в диапазоне от 0 до 255.

Логический 32-битный IP-адрес представляет собой иерархическую систему и состоит из двух частей. Первая идентифицирует сеть, вторая - узел в сети. Обе части являются обязательными.

Например, если IP-адрес узла – 192.168.18.57, то первые три октета (192.168.18) представляют собой сетевую часть адреса, а последний октет (.57) является идентификатором узла. Такая система называется иерархической адресацией, поскольку сетевая часть идентифицирует сеть, в которой находятся все уникальные адреса узлов. Маршрутизаторам нужно знать только путь к каждой сети, а не расположение отдельных узлов.

Другой пример иерархической сети – это телефонная сеть. В телефонном номере код страны, региона и станции составляют адрес сети, а оставшиеся цифры - локальный номер телефона.

Каждый IP-адрес состоит из двух частей. Как узлы определяют, где сетевая часть, а где адрес узла? Для этого используется **маска подсети.**

При настройке IP узлу присваивается не только IP-адрес, но и маска подсети. Как и IP-адрес, маска состоит из 32 бит. **Маска подсети определяет, какая часть IP-адреса относится к сети, а какая – к узлу.**

Маска сравнивается с IP-адресом побитно, слева направо. В маске подсети единицы соответствуют сетевой части, а нули - адресу узла. В приведенном примере первые три октета представляют собой адрес сети, а последний - адрес узла.

Отправляя пакет, узел сравнивает маску подсети со своим IP-адресом и адресом назначения. Если биты сетевой части совпадают, значит, узлы источника и назначения находятся в одной и той же сети, и пакет доставляется локально. Если нет, отправляющий узел передает пакет на интерфейс локального маршрутизатора для отправки в другую сеть.

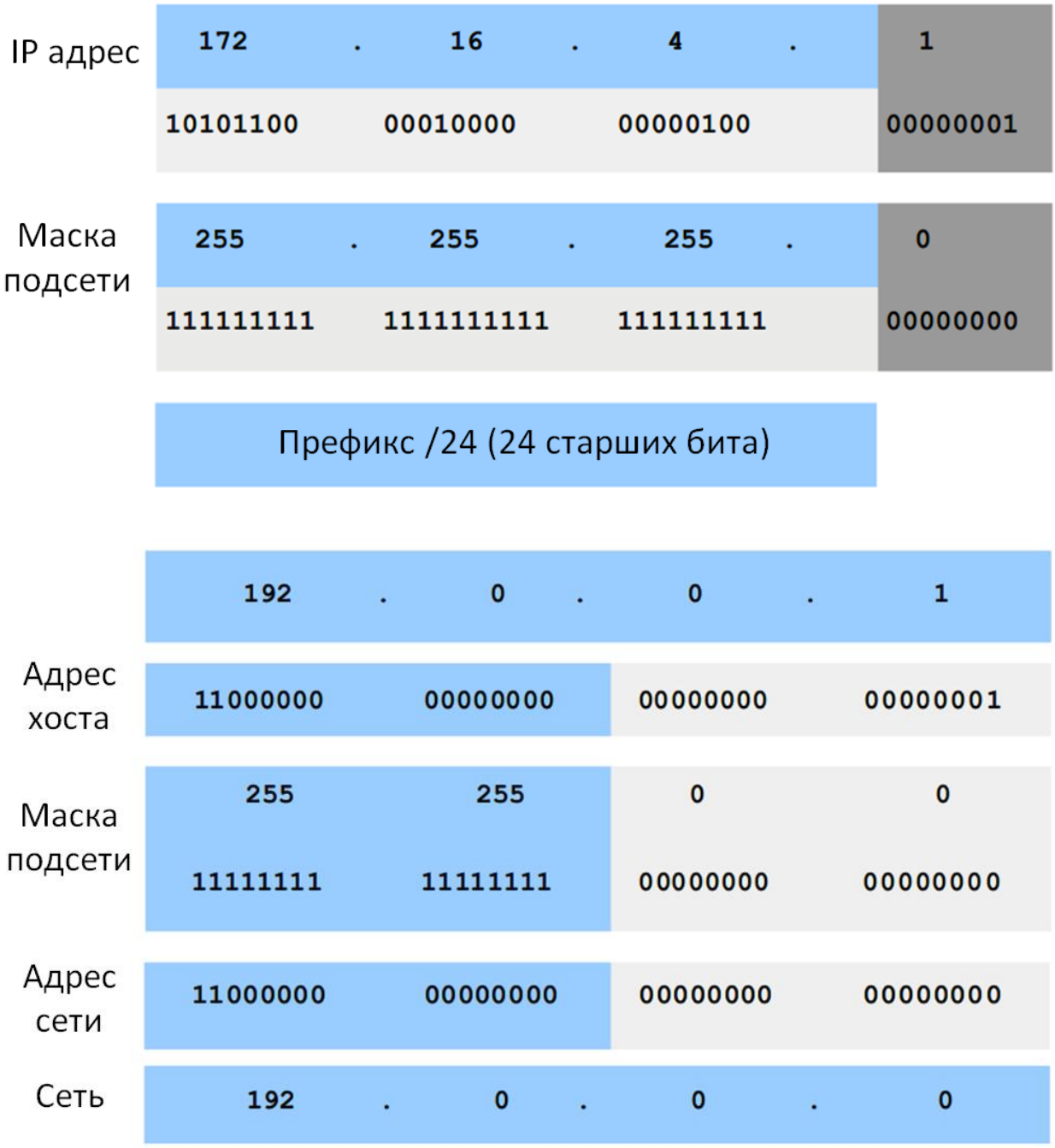


Рисунок 5.31 – Пример вычисления двух разных масок подсети

В домашних офисах и небольших компаниях чаще всего встречаются следующие маски подсети: 255.0.0.0 (8 бит), 255.255.0.0 (16 бит) и 255.255.255.0 (24 бита). В маске подсети 255.255.255.0 (десятичный вариант), или 11111111.11111111.1111111.00000000 (двоичный вариант) 24 бита идентифицируют сеть, а 8 - узлы в сети.

Чтобы вычислить количество возможных сетевых узлов, нужно взять количество отведенных для них бит в степени 2 ($2^8 = 256$). Из полученного результата необходимо вычесть 2 (256-2). Дело в том, что состоящая из одних единиц (1) отведенная узлам часть IP-адреса предназначена для адреса широковещательной рассылки и не может принадлежать одному узлу. Часть,

состоящая только из нулей, является идентификатором сети и тоже не может быть присвоена конкретному узлу[5].

В 16-битной маске для адресов узлов отводится 16 бит (два октета), и в одном из них все значения могут быть равны 1 (255). Это может быть и адрес широковещательной рассылки, но если другой октет не состоит из одних единиц, адрес можно использовать для узла. Не забывайте, что узел проверяет значения всех бит, а не значения одного октета.

IP-адрес и маска подсети совместно определяют то, какая часть IP-адреса является сетевой, а какая - соответствует адресу узла.

IP-адреса делятся на 5 классов. К классам A, B и C относятся коммерческие адреса, присваиваемые узлам. Класс D зарезервирован для многоадресных рассылок, а класс E – для экспериментов.

В адресах класса C сетевая часть состоит из трех октетов, а адрес узла – из одного. Выбранная по умолчанию маска подсети состоит из 24 бит (255.255.255.0). Адреса класса C обычно присваиваются небольшим сетям.

В адресах класса B сетевая часть и адрес узла состоят из двух октетов. Выбранная по умолчанию маска подсети состоит из 16 бит (255.255.0.0). Обычно эти адреса используются в сетях среднего размера.

В адресах класса A сетевая часть состоит всего из одного октета, остальные отведены узлам. Выбранная по умолчанию маска подсети состоит из 8 бит (255.0.0.0). Обычно такие адреса присваиваются крупным организациям.

Класс	Диапазон значений 1 октета	Биты первого октета	Части адресов сети (N) и хоста (H)	Маска подсети по умолчанию	Число подсетей и хостов
A	1-127	00000000 – 01111111	N.H.H.H	255.0.0.0	128 сетей (2^7) 16777214 хостов в сети(2^{24}-2)
B	128-191	10000000 – 10111111	N.N.H.H	255.255.0.0	16 384 сетей (2^{14}) 65 534 хостов в сети (2^{16} - 2)
C	192-223	11000000 – 11011111	N.N.N.H	255.255.255.0	2 097 150 сетей (2^{21}) 254 хоста в сети (2^8 - 2)
D	224-239	11100000 – 11101111	Мультикастовая адресация		
E	240-255	11110000 – 11111111	Экспериментальная адресация		

Адрес 127.0.0.1 – «локальная петля», локальный IP-адрес по-умолчанию

Рисунок 5.32 – Классы IPv4-адресов

[5] Иначе допустимое количество узлов можно определить, сложив значения доступных бит (128+64+32+16+8+4+2+1 = 255). Из полученного значения необходимо вычесть 1 (255-1 = 254), поскольку значение всех бит отведенной для узлов части не может равняться 1. Число 2 вычитать не нужно, поскольку сумма нулей равна нулю и в сложении не участвует.

Класс адреса можно определить по значению первого октета. Например, если значение первого октета IP-адреса находится в диапазоне от 192 до 223, то это адрес класса C. Например, адрес 200.14.193.67 относится к классу C.

Всем узлам, подключенным непосредственно к Интернету, необходим уникальный публичный IP-адрес. Поскольку количество 32-битных адресов конечно, существует риск, что их не хватит. В качестве одного из решений было предложено зарезервировать некоторое количество частных адресов для использования только внутри организации. В этом случае внутренние узлы смогут обмениваться данными друг с другом без использования уникальных публичных IP-адресов.

В соответствии со стандартом RFC 1918 было зарезервировано несколько диапазонов адресов класса A, B и C.

Таблица 5.2 – Зарезервированное пространство частных адресов

Класс адреса	**Число зарезервированных адресов**	**Диапазон адресов**
A	1	10.0.0.0
B	16	172.16.0.0 – 172.31.0.0
C	256	192.168.0.0 – 192.168.255.0

В диапазон частных адресов входит одна сеть класса A, 16 сетей класса B и 256 сетей класса C. Таким образом, сетевые администраторы получили определенную степень свободы в плане предоставления внутренних адресов.

В очень большой сети можно использовать частную сеть класса A, где можно создать более 16 миллионов частных адресов.

В сетях среднего размера можно использовать частную сеть класса B с более чем 65 000 адресов.

В домашних и небольших коммерческих сетях обычно используется один частный адрес класса C, рассчитанный на 254 узла.

Одну сеть класса A, 16 сетей класса B или 256 сетей класса C могут использовать организации любого размера. Многие организации пользуются частной сетью класса A.

Узлы из внутренней сети организации могут использовать частные адреса до тех пор, пока им не понадобится прямой выход в Интернет. Соответственно, один и тот же набор адресов подходит для нескольких организаций. Частные адреса не маршрутизируются в Интернете и быстро блокируются маршрутизатором поставщика услуг Интернета.

Частные адреса можно использовать как меру безопасности, поскольку они видны только в локальной сети, а посторонние получить прямой доступ к этим адресам не могут.

Кроме того, существуют частные адреса для диагностики устройств. Они называются адресами обратной связи (loopback, локальная петля). Для таких адресов зарезервирована сеть 127.0.0.0 класса A.

5.6 Сеть INTERNET

Всемирная паутина (World Wide Web - WWW), компьютерная сеть информационных ресурсов, через которую пользователь может двигаться, используя связи одного документа с другими. Информация по Всемирной паутине распространяется по компьютерам всего мира. Всемирная паутина часто упоминается просто как «Сеть» (Web).

Сеть стала очень популярным информационным ресурсом с тех пор, как впервые стало возможным представлять изображения и другие мультимедиа продукты в Internet, всемирной сети компьютеров, в 1993 г. Сеть предлагает место, где компании, учреждения, и личности могут отображать информацию относительно их изделий, программ, исследований или их жизней. Сеть стала форумом для многих групп и рынком для многих компаний. Музеи, библиотеки, правительственные агентства и школы считают Сеть ценнейшим изобретением, она также несет информацию в широком спектре форматов.

Подобно всем сетям ЭВМ, Web объединяет два типа компьютеров - клиентов и серверов - с использованием стандартного набора правил (протокола) для связи между компьютерами. Компьютер-сервер содержит информационные ресурсы, которые содержатся в Сети, и пользователи Сети используют компьютеры-клиенты, чтобы обратиться к ресурсам. Компьютерная сеть может быть сетью общего пользования типа всемирного Internet или частной сетью, типа Intranet компании. Web - часть Internet, Internet также включает и другие средства межкомпьютерного обмена, типа Telnet, протокола передачи файлов FTP и т.д., но Web быстро стала наиболее широко используемой частью Internet. Она отличается от других частей Internet правилами, которые компьютеры используют для общения друг с другом, и доступностью иной, чем текст, информация. Намного труднее иметь дело с изображениями или другими мультимедиа-файлами иными методами, чем применяемыми в Web.

Предоставление компьютером-клиентом возможности отобразить страницы сети с изображениями и другими медиа-средствами стало возможным после введения специального программного продукта, называемого браузером (от англ. Browse - просматривать). Каждый документ Сети содержит кодированную информацию относительно того, что находится на странице, как страница должна просматриваться и с какими другими сайтами (информационными узлами) документ связан. Программа просмотра на компьютере пользователя читает эту информацию и использует ее, чтобы отобразить страницу на экране пользователя. Почти каждая Страница сети или документ Сети включают связи, названные гиперссылками (hyperlinks), с другими сайтами. Гиперссылки - определяющая особенность Сети - они позволяют пользователям путешествовать между документами Сети без следования специальному порядку или иерархии.

Когда пользователи хотят обратиться к Сети, они используют браузер Сети на их компьютере-клиенте, чтобы соединиться с компьютером-сервером Сети. Компьютеры-клиенты соединяются с Сетью одним из двух способов. Клиенты с разрешенным доступом подключаются либо прямо в Сеть посредством маршрутизатора (роутера, router) либо с помощью локальной сети, прямо подключенной к Сети. Клиенты с удаленным доступом соединяются с Сетью посредством модема (modem, от МОдулятор-ДЕМодулятор), аппаратного устройства, которое транслирует информацию от компьютера в сигналы, которые могут передаваться по телефонным линиям или сетям цифрового телевидения. Некоторые модемы посылают сигналы по каналам кабельного телевидения. Разумеется, существуют варианты подключения беспроводным способом, используя сеть WiFi.

Серверы Сети содержат документы и мультимедиа-средства, связанные с ними. Они могут быть обыкновенными персональными компьютерами, мощными универсальными компьютерами или чем-то промежуточным между ними. Клиентом может быть любой тип компьютера, от классической рабочей станции до смартфона. Web и все форматы Internet используют набор (иногда говорят - стек) протоколов, названный TCP/IP. Однако, каждая часть Internet использует несколько различные системы для передачи файлов между клиентами и серверами.

Адрес документа Сети помогает компьютеру пользователя найти и соединиться с сервером, который содержит нужную страницу. Адрес страницы сети называется URL (Uniform Resource Locator).

URL - составной код, который сообщает браузеру клиента три вещи:

- правила (протокол), которые пользователь должен использовать, чтобы получить доступ к сайту;
- адрес Internet, который уникально определяет сервер;
- расположение в пределах файловой системы сервера данного элемента.

Пример: URL – http://encarta.msn.com.

Первая часть, URL, http://, показывает, что сайт находится во Всемирной паутине. Большинство браузеров также способно к воспроизведению файлов с форматами других частей Internet типа FTP. Другие форматы Internet используют различные коды первой части их URL - например, FTP использует ftp://.

Следующая часть URL, encarta.msn.com, дает название или уникальный адрес в Internet сервера, на котором хранится сайт.

Некоторые URL определяют специфические каталоги, или файлы такие, как http://encarta.msn.com/explore/default.asp-explore является названием каталога, в котором находится файл default.asp.

Необходимо упомянуть о системе символьных адресов в Сети - DNS.

Доменная система имен (Domain Name System, DNS) – это распределенная база данных, которая содержит информацию о компьютерах (хостах), включенных в сеть Internet. Чаще всего информация включает имя машины, IP-адрес и данные для маршрутизации почты.

Как известно, для обращения к хостам в сети Internet используются 32-разрядные IP-адреса, однозначно идентифицирующие любой сетевой компьютер в этой сети. Однако для пользователей применение IP-адресов при обращении к хостам не удобно. Поэтому была создана система преобразования имен, позволяющая компьютеру в случае отсутствия у него информации о соответствии имен и IP-адресов получить необходимые сведения от DNS-сервера, ip-адрес которого хранится в настройках подключения к Internet.

Основная задача DNS — преобразование имен компьютеров в IP-адреса и наоборот.

Для реализации системы DNS был создан специальный сетевой протокол DNS. В сети имеются специальные выделенные информационно-поисковые серверы - DNS-серверы.

Пространство имен DNS имеет вид дерева доменов с полномочиями, возрастающими по мере приближения к корню дерева.

Рисунок 5.33 – Пример пространства имен DNS

По историческим причинам существует два вида имен доменов верхнего уровня. В США домены верхнего уровня отражают организационно-политическую структуру и, как правило, имеют трехбуквенные имена. Для доменов вне США используются двухбуквенные коды стран ISO. Оба эти принципа сосуществуют в одном глобальном пространстве имен. Имена доменов верхнего уровня на текущий момент времени перечислены в таблице 5.3. Некоторые коды стран приведены в таблице 5.3. В именах доменов не учитывается регистр.

Таблица 5.3 – Примеры доменов первого уровня

Географические	Тематические
.RU – Россия .KZ – Казахстан .BY – Белоруссия .SU – СССР .FR – Франция .DE – Германия	.COM – коммерческие ресурсы .NET – ресурсы, связанные с сетью .ORG – некоммерческие организации .INFO – информационные ресурсы .EDU – образовательные организации .GOV – правительственные организации

Доменами второго уровня централизованно управляет Информационный центр сети (Network Information Center, NIC).

На некоторые имена наложено табу; это, в частности, относится к уже взятым именам, ключевому слову AT, комбинациям имен доменов верхнего уровня (например, edu.com) и повторениям имен (например, x.x.com).

Документ RFC1032 рекомендует, чтобы имена доменов второго уровня имели в длину не более 12 символов, несмотря на то, что DNS допускает использовать в каждой составляющей до 64 символов.

Продолжим краткий экскурс в функционал сети Internet. Сеть содержит информацию во многих формах, включая текст, графические изображения и любой тип цифрового медиа-файла, включая видео- и звуковые файлы. Некоторые элементы страниц сети фактически являются небольшими программами с их собственными правами. Эти объекты, называемые апплетами (от небольшого приложения - другое название для компьютерной программы), следуют определенному набору команд.

Апплеты позволяют пользователям запустить в Сети игры, поиск в базах данных, выполнять виртуальные научные эксперименты и множество других действий.

Коды, которые сообщают браузеру на компьютере пользователя, как отобразить документ Сети, соответствуют своду правил, названному Языком разметки гипертекста (HTML – HyperText Markup Language).

Каждый документ Сети написан как открытый текст, и команды, которые сообщают компьютеру пользователя, как представить документ, содержатся в самом документе непосредственно, закодированными с использованием специальных символов, названных тэгами (tag) HTML. Браузер знает, как интерпретировать тэги HTML, так что документ появляется на пользовательском экране именно так, как имел в виду проектировщик документа (также называемый веб-дизайнером, web-designer).

В дополнение к HTML, некоторые типы объектов в Сети используют свое собственное кодирование. Апплеты, например, являются мини-программами, которые написаны на языках программирования типа Visual Basic и Java.

Клиент-серверная связь, URL и HTML позволяют информационным узлам (сайтам, хостам) включать гиперссылки, которые пользователи могут использовать, чтобы путешествовать «сквозь» Сеть. Гиперссылки часто являются фразами в тексте документа Сети, которые связываются с другим документом Сети, снабженным своим URL, когда пользователь щелкает своей мышью на этой фразе. Браузер пользователя обычно различает гиперссылки и обычный текст, помечая гиперссылки различным цветом или подчеркиванием. Гиперссылки позволяют пользователям переходить между разбросанными на Сети страницами не в каком-то определенном порядке. Этот метод доступа к информации назван ассоциативным доступом, и ученые уверяют, что это подобие того, каким путем человеческий мозг получает доступ к хранящейся информации. Гиперссылки делают ссылочную информацию в Сети быстрее и проще, чем при использовании традиционных печатных документов.

Один из наиболее замечательных аспектов Всемирной паутины - ее пользователи. Они - поперечный разрез общества. Пользователи включают студентов, которые должны найти материалы на заданную тему, врачей, которые нуждаются в информации относительно самого последнего медицинского исследования, и абитуриентов колледжа, исследующих университетские городки или интерактивно заполняющих заявки о финансовой помощи. Другие пользователи включают инвесторов, которые могут интересоваться деловой историей акционерной компании и оценить данные относительно различных общественных и открытых фондов. Вся эта информация с готовностью располагается в Сети. Пользователи могут часто находить финансовые диаграммы о деятельности компании, которые представляют информацию самыми различными способами.

Путешественники, изучающие возможную поездку, могут совершать виртуальные туры, посмотреть расписания авиалиний и плату за проезд или купить билет на рейс с помощью Сети. Большинство мест назначения, включая парки, города, гостиницы - имеют их собственные сайты с руководствами и местными картами. Крупные компании – поставщики товаров также имеют информационные узлы, на которых заказчики могут проследить процесс отгрузки, выяснить, где их товары находятся или когда они будут поставлены (этот процесс часто называется трекинг).

Правительственные агентства имеют информационные узлы, где они отправляют по почте инструкции, процедуры, информационные бюллетени и налоговые формы. Многие должностные лица имеют свои сайты, где они выражают свои взгляды, перечисляют собственные достижения и т.п. Сеть также содержит каталоги почтовых адресов, электронной почты и номеров телефонов.

Пользователи Сети могут посетить сайты крупных книжных магазинов,

одежды и других товаров. Многие центральные газеты имеют специальные электронные издания, которые выпускаются чаще, чем ежедневно. Электронные журналы почти в каждой отрасли науки – теперь в Сети. Большинство музеев предлагает пользователю виртуальный тур по их экспозициям и зданиям. Эти организации и учреждения обычно используют сайты, чтобы дополнить неэлектронные части деятельности. Некоторые получают дополнительные доходы от продажи места для публикации рекламных объявлений на своих сайтах.

Всемирная паутина была разработана британским физиком и компьютерным специалистом Тимоти Бернерсом-Ли как проект в рамках исследований для Европейского Центра Ядерной энергии (CERN, теперь Европейская Лаборатория Физики элементарных частиц) в Женеве, Швейцария. Бернес-Ли первым начал работать с гипертекстом в начале 1980-ых гг. Созданная им Сеть стала функционировать в CERN в 1989 году, и затем стала быстро распространяться по университетам в остальной части мира с помощью ученых-ядерщиков. Группы в Национальном Центре Прикладных программ Супервычислений в Университете Штата Иллинойс также исследовали и разработали технологию Сети. Они первыми разработали браузер, названный Мозаика (Mosaic), в 1993 г.

Итак, с конца 1960-х до начала 1990-х годов Internet был инструментом связи и исследований, используемым почти исключительно для академических и военных целей. Это положение изменилось радикально с введением Всемирной паутины (также называемой WWW, или W3) в 1989 г.

WWW - набор программ, стандартов и протоколов, с помощью которых мультимедиа-файлы (документы, которые могут содержать текст, фотографии, графику, видео и звук) создаются и отображаются в Internet.

Internet включает WWW, а также включает аппаратные средства (компьютеры, супер-ЭВМ и связи) и не WWW-программное обеспечение и протоколы, на которых WWW выполняется. Различие между Internet и WWW подобно различию между компьютером и программой мультимедиа, которая выполняется на компьютере. Всплеск популярности Internet в 1990-х наиболее вероятен из-за интенсивного применения графики во Всемирной паутине.

Наиболее широко используемый инструмент в Internet – электронная почта или e-mail. Электронная почта используется, чтобы посылать письменные сообщения между отдельными лицами или группами лиц, часто географически разделенных большими расстояниями. Сообщения электронной почты обычно посылаются и принимаются почтовыми серверами - компьютерами, которые специализированы для обработки и отправления электронной почты. Как только сервер получил сообщение, он направляет его на компьютер, которому данная почта адресована.

До введения World Wide Web существовали различные стандарты и типы программного обеспечения для передачи данных по Internet. Многие из них все еще используются, например FTP. File Transfer Protocol - протокол передачи файлов является методом перемещения файлов от одного

компьютера до другого по Internet, даже если каждый компьютер имеет различную операционную систему или формат хранения данных.

В то время как эти протоколы передачи и программное обеспечение все еще используются, WWW намного более легка для применения и используется намного чаще, чем более ранние протоколы передачи.

Главная проблема, возникшая в процессе длительного роста Internet - трудность обеспечения достаточной ширины полосы передачи, чтобы поддерживать функционирование Сети. Поскольку приложения Internet становятся все более сложными, и поскольку все большее количество людей во всем мире использует Internet, количество информации, передаваемой через Internet, будет требовать связи с очень большой шириной полосы передачи. В то время как многие телекоммуникационные компании пытаются разрабатывать более производительные технологии, не известно, будут ли эти технологии способны удовлетворить растущий спрос.

5.7 Протокол IPv6. Проблемы и перспективы развития Internet-адресации

Скажем еще несколько слово о протоколе IPv4 (RFC-791), на котором в данный момент основан Internet почти полностью, т.к. IPv4 является основной частью стека TCP/IP.

Этот протокол занимается маршрутизацией в сетях, т.е. он направляет пакет по пути от отправителя до получателя. IP-протокол посылает данные дейтограммами. Каждая такая дейтаграмма, кроме передаваемых данных, содержит в себе и заголовок. Обычно заголовок содержит 20 октетов, т.е. имеет длину 20 байт, но эта длина может варьироваться, что отнюдь не упрощает процесс передачи данных.

Наиболее важными в рамках обсуждаемого вопроса полями заголовка являются поля IP-адресов (адрес отправителя и получателя), поле тип сервиса (изначально неиспользовавшееся, а теперь устаревшее) и поле протокол (определяющее структуру поля "данные", т.е. в соответствии с каким протоколом они кодировались).

Итак, перечислим общие недостатки протокола IPv4:

- дефицит адресного пространства: количество различных устройств, подключаемых к сети Internet, растет экспоненциально, размер адресного пространства 2^{32} быстро истощается;
- слабая расширяемость протокола: недостаточный размер заголовка IPv4, не позволяющий разместить требуемое количество дополнительных параметров в нем;
- проблема безопасности коммуникаций: не предусмотрено каких-либо средств для разграничения доступа к информации, размещенной в сети.
- отсутствие поддержки качества обслуживания: не поддерживается размещение информации о пропускной способности, задержках, требуемой для нормальной работы некоторых сетевых приложений;
- проблемы, связанные с механизмом фрагментации: не определяется

размер максимального блока передачи данных по каждому конкретному пути;

- отсутствие механизма автоматической конфигурации адресов;
- проблема перенумерации машин.

Поля IP-адресов содержат IP-адреса отправителя и получателя. В IPv4 IP-адрес состоит из 4 байт и часто представляется в виде 4 чисел, размером 1 байт, разделяемые точками, что даёт чуть больше 4 миллиардов различных адресов. В такой схеме каждый компьютер в Internet имеет свой уникальный адрес. Но при появлении Internet-ресурсов адреса распределялись огромными блоками. Так Массачусетский университет имеет у себя блок в 16 миллионов адресов, в то время, когда средний по величине провайдер имеет несколько тысяч адресов(при намного (!) меньшем количестве линий связи). Такое "растранжиривание" IP-адресов привело к тому, что их пространство начало заканчиваться и ощущаться нехватка. Фактически в 2012 году 32-битное пространство адресов исчерпало себя, что привело к некоторой задержке в развитии Internet. Хотя и были предприняты меры решения этой проблемы (например технологии выделения блоков по 2 адреса и технология раздачи динамических IP-адресов DHCP, а так же NAT, позволяющая транслировать IP-адреса из внешней сети во внутреннюю), всё равно, "финал" удалось лишь отсрочить. При нынешних темпах роста Internet, согласно IPv4 Address Space Report (bgp.potaroo.net/ipv4), не розданные пока адреса закончатся к 2018 году, а 32-битное IP-пространство абсолютно полностью исчерпается в 2040 году (хотя есть и намного менее оптимистичные прогнозы). Проблема адресного пространства IPv4 считается основной, но она не главная и не единственная.

Проблемы масштабируемости IPv4 на этом не заканчиваются. Есть также проблемы связанные с транспортировкой данных, заключающиеся в сложности маршрутизации и, следовательно, разрастании таблиц маршрутизации. Хоть эту проблему и пытались решить введением бесклассовой междоменной маршрутизацией (CIDR), всё равно она этим не разрешилась, зато IP-администрирование от этого усложнилось (всё же трудно построить и упорядочить структуру адресов в 32-битном пространстве). Также существует сложность обработки IP-заголовка. Существенное число полей, которое он содержит, отнюдь не облегчает жизнь процессам, обрабатывающим такие заголовки, ведь некоторые поля заголовка уже потеряли свою значимость, а другие возможно существенно упростить. Также переменная длина заголовка заставляет постоянно пересчитывать контрольную сумму, а при высоких скоростях это сильно загрузит процессор.

Также, в IPv4 отсутствуют некоторые механизмы, необходимые по современным меркам. Это механизмы информационной безопасности и средства поддержки классов обслуживания. Отсутствуют методы шифрования данных, которые сейчас на практике очень пригождаются. А такие средства должны быть реализованы именно на сетевом уровне, чтобы не переносить эти задачи этим приложения. Обеспечить поддержку классов

обслуживания должны опять же маршрутизаторы, связывающие системы, чтобы эта задача не ставилась на уровне приложений, что опять же приведёт к усложнению и нестабильности работы.

Реализовывать эти механизмы на IPv4 дополнительно - пустая трата времени. Всё равно придётся для этого вносить изменения в весь стек TCP/IP, что в любом случае приведёт к некоторому усложнению работы, связанному с переходом на новый стандарт.

В итоге, можно сказать, что IPv4 отслужил почти 30 лет верой и правдой и мог бы служить дальше, но не стоит терпеть устаревший протокол, содержащий неисправимые на сетевом уровне недостатки, в то время, когда на смену ему давно готова отличная альтернатива - IPv6.

История IPv6 (RFC-2460) начинается с 1992 года. Тогда он был разработан для решения проблем адресного пространства и ряда смежных задач. Решено, что адресное пространство IPv6 будет распределяться IANA (Internet Assigned Numbers Authority - комиссия по стандартным числам в Internet [RFC-1881]), которая будет иметь региональных представителей, которые будут подробно заниматься выдачей IP-адресов в своих областях. Такое распределение не будет необратимым. IANA сможет в любой момент перераспределить адресное пространство, в случае допущения ошибок при его распределении. Иными словами, все сделано так, чтобы не повторить прежних ошибок IPv4.

Что касается самого адресного пространства, то оно будет расширено с прежних 4 миллиардов с небольшим IPv4 до 340 282 366 920 938 463 463 374 607 431 768 211 456 адресов! IP-пространство IPv6 будет 128-битным, что добавит возможностей маршрутизации. Адреса IPv6 также способны бесконтекстно автоконфигурироваться. Такие адреса существенно упростят маршрутизацию и сократят таблицы маршрутизации в несколько раз.

Кроме явного преимущества в расширении адресного пространства, можно выделить следующие преимущества IPv6 над IPv4:

- Возможность автоконфигурирования IP адресов;
- Упрощение маршрутизации;
- Облегчение (упрощение) заголовка пакета;
- Поддержка качества обслуживания (QoS);
- Наличие возможности криптозащиты датаграмм на уровне протокола;
- Повышенная безопасность передачи данных.

Собственно, почти все преимущества IPv6 вытекают как раз из формата его пакета и формы адресации. Переделанный и усовершенствованный стандарт позволить реализовать на уровне протокола мощную криптозащиту (шифрование данных) и многие сервисы, такие как QoS (Quality of Service). QoS в IPv6 поддерживается полностью на сетевом уровне, это крайне важно для мультимедиа-трансляций. Изменения, внесённые в IPv6 показывают, что он не просто решит основную проблему нехватка адресного пространства, а перестроит всю структуру Internet так, что она станет более логичной и продуманной.

Также, в новом протоколе будет возможность автоконфигурирования IP адресов для конечных компьютеров в сети двумя способами: с помощью усовершенствованного DHCP или без него.

В протоколе IPv6 пакеты не могут фрагментироваться и собираться маршрутизаторами. Отправитель обязан заранее выяснить максимальный размер пакетов (Maximum Transmission Unit, MTU), поддерживаемый на всём пути до получателя, и, при необходимости, выполнить фрагментацию своими силами. Снятие с маршрутизаторов забот о фрагментации также способствует повышению эффективности их работы, хотя и немного усложняет в определённой степени работу и функциональность оконечных систем.

Также, создатели заверяют, что с приходом этого протокола будет повышена сетевая безопасность: хакерам будет невозможно проводить DoS атаки (или закидывания пингами) и сканировать сети.

Переход на IPv6 неизбежен в любом случае. Но идёт он медленно по причине того, что польза от нововведений не столь очевидна на данный момент для большинства пользователей. В основном первыми переходят те страны или районы, где недостаток адресов ощущается наиболее остро.

Контрольные вопросы

1. Что такое локальная вычислительная сеть?
2. Перечислите семь уровней взаимодействия открытых систем.
3. Какие топологии вычислительных сетей вы знаете? В чем суть каждой из них?
4. В чем заключается главный недостаток кольцевой топологии?
5. Какие виды кабелей используются при формировании сетей?
6. В чем суть метода передачи информации в сети Token Ring?
7. Как формируется сеть Ethernet?
8. Опишите принципы IPv4-адресации
9. Что такое Internet?
10. Опишите проблемы Internet-адресации и смысл IPv6-адресации

ЗАКЛЮЧЕНИЕ

Учебное пособие позволяет студентам-первокурсникам получить начальную базовую подготовку в области информационных систем и технологий. Последующие дисциплины технического направления углубляют изложенные в учебном пособии основы информатики и информационных технологий и дают возможность выполнения на практике основных видов деятельности специалиста с высшим образованием в данной области.

Материал учебного пособия позволяет лучше познакомится с принципами организации и работы таких объектов профессиональной деятельности бакалавров, как вычислительные машины, комплексы, системы и сети; автоматизированные системы обработки информации и управления; системы автоматизированного проектирования и информационной поддержки жизненного цикла промышленных изделий; программное обеспечение средств вычислительной техники и автоматизированных систем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Основная литература

1. Савельев А.Я. Основы информатики: Учеб. Для вузов. – М.: Изд-во МГТУ им. Н.Э.Баумана, 2001

2. Вирт Н. Алгоритмы и структура данных. - М.: Мир, 1989.

3. Першиков В.И., Савинов В.М. Толковый словарь по информатике. - М.: Финансы и статистика, 1991.

4. Власов В.К., Королев Л.Н., Сотников А.Н. Элементы информатики /Под ред Л.Н. Королева. - М.: Наука, Гл. ред. физ.-мат. лит., 1988.

5. Информатика. Энциклопедический словарь для начинающих. - М.: Педагогика-Пресс, 1994.

6.Острейковский В.А. Информатика: Учеб. Для вузов. – М.: Высш. шк., 2001.

7. Бородакий Ю.В., Лободинский Ю.Г. Информационные технологии. Методы, процессы, системы – М.: Радио и связь, 2002.

8. Алферова Т.В. Теория алгоритмов. - М: Статистика, 1973.

9. Дж. фон-Нейман. Теория самонастраивающихся автоматов. - М.: Мир, 1971.

10. Поспелов Д. А. Арифметические основы вычислительных машин дискретного действия. - М.:Энергия, 1970.

11. Савельев А. Я. Арифметические и логические основы цифровых автоматов. - М.: Высшая школа, 1980.

12. Темников Ф.Е., Афонин В.А., Дмитриев В.И. Теоретические основы информационной техники. - М.: Высшая школа, 1979.

Дополнительная литература

13. Вопросы прикладной информатики (сборник научных трудов) /Под ред Р.М. Юсупова - С.-Пб: СПИИРАН, 1993.

14. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. - М.: Мир, 1981.

15. Жоголев Е.А., Трифонов Н.П. Курс программирования. - М.: Наука, 1967.

16. Информатика. Терминологический словарь. - М.: Всероссийский НИИ комплексной информации по стандартизации и качеству, 1992.

17. Каныгин Ю.М., Калитич Г.И. Основы теоретической информатики. - Киев: Наукова думка, 1990.

18. Мепер Б., Бодуэн К. Методы программирования. - М.: Мир, 1982.

19. Романов Е.Л. Си/С++. От дилетанта до профессионала [электронный ресурс] – http://ermak.cs.nstu.ru/cprog/HTML/index.htm.

20. Горнец Н.Н., Рощин А.Г., Соломенцев В.В. Организация ЭВМ и систем: учебное пособие для студентов вузов – М.:Академия, 2006.

ПРИЛОЖЕНИЕ

Введение в математическую логику

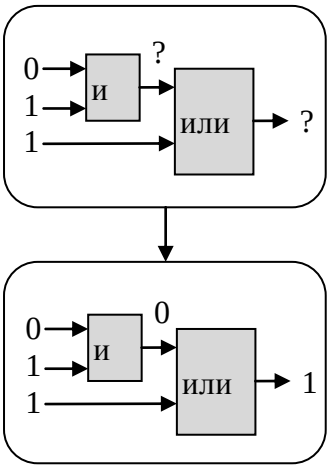
Логическая константа	Логическая величина	Логическая переменная
(фиксированное значение)	(понятие, выражаемое словами)	(символически обозначенная логическая величина, которая может принимать любые значения)
ИСТИНА или ЛОЖЬ	ИСТИНА (true) ЛОЖЬ (false)	ИСТИНА или ЛОЖЬ

Высказывание - это повествовательное предложение, в котором что-либо утверждается или отрицается. По поводу любого высказывания можно сказать, истинно оно или ложно.

Логическое высказывание - это простое или сложное высказывание, представленное в символической форме. Сложное высказывание строится из простых с помощью логических операций (связок).

Логические схемы

1 - истина 0 - ложь

Конъюнкция	Дизъюнкция	Инверсия	Пример: 0 и 1 или 1
0, 0 → И → 0	0, 0 → ИЛИ → 0	0 → НЕ → 1	
0, 1 → И → 0	0, 1 → ИЛИ → 1		
1, 0 → И → 0	1, 0 → ИЛИ → 1	1 → НЕ → 0	
1, 1 → И → 1	1, 1 → ИЛИ → 1		

Приоритеты при выполнении логических операций

НЕ → И → ИЛИ

Основные логические операции

"И" (AND)

Конъюнкция
логическое умножение

$F = a \wedge b$

a и b → F

a	**b**	**F**
0	0	0
0	1	0
1	0	0
1	1	1

"ИЛИ" (OR)

Дизъюнкция
логическое сложение

$F = a \vee b$

a или b → F

a	**b**	**F**
0	0	0
0	1	1
1	0	1
1	1	1

"НЕ" (NOT)

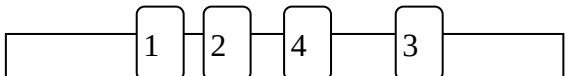
Инверсия
логическое отрицание

$F = \overline{a}$

не a → F

a	**b**
0	0

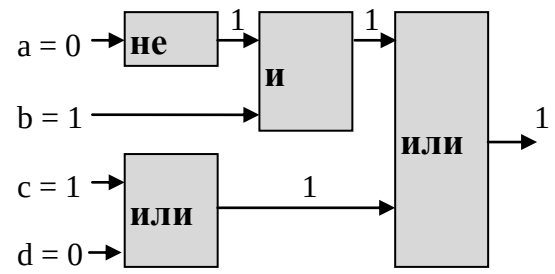
Примеры вычисления логических выражений

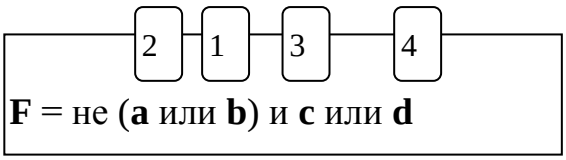


F = не **a** и **b** или (**c** или **d**)

a	b	c	d
0	1	1	0

1	не **0** → **1**
2	**1** и **1** → **1**
3	**1** или **0** → **1**
4	**1** или **1** → **1**

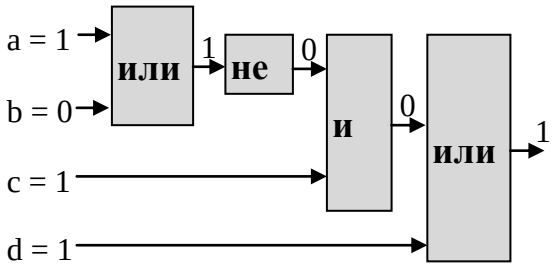




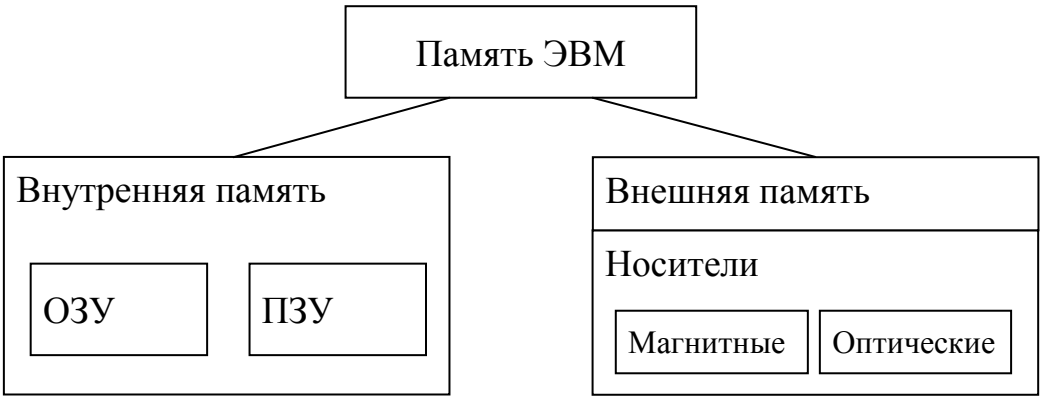
F = не (**a** или **b**) и **c** или **d**

a	b	c	d
1	0	1	1

1	**1** или **0** → **1**
2	не **1** → **0**
3	**0** и **1** → **0**
4	**0** или **1** → **1**



Память ЭВМ



ОЗУ - оперативное запоминающее устройство
(энергозависимая память для чтения и записи информации)

ПЗУ - постоянное запоминающее устройство
(энергонезависимая память только для чтения информации)

Единицы памяти:
бит, байт (8 бит), машинное слово (1, 2, 4, ... байта)

Числа в памяти ЭВМ

Форма с фиксированной точкой
Целые числа

Целое число в памяти ЭВМ занимает машинное слово

Старший разряд в представлении числа является знаковым:
0 - положительное число
1 - отрицательное число

Отрицательные числа в памяти ЭВМ представлены в виде дополнительного кода, что дает возможность заменить операцию вычитания операцией сложения с отрицательным числом
N - M = N + (-M)

Форма с фиксированной точкой
Целые числа

$R = \pm m \times p^n$,
где: m - мантисса
p - основание системы счисления
n - порядок

$0{,}25324 \times 10^2 = 0{,}025324 \times 10^3 = 2{,}5324 \times 10^4$

Нормализованная форма
$0{,}1 \le m < 1$

Представление вещественного числа в памяти ЭВМ в виде пары целых чисел: матиссы (m) и порядка (n)
Пример: m = 25324, n = 2

Один из вариантов представления вещественного числа в 4-байтовой ячейке памяти

±	Порядок	М а н т и с с а	

Внутренняя память ЭВМ

0 или **1**
Двоичная кодировка

Структура внутренней памяти

Байты	Биты
0	0 1 1 0 0 1 0 1
1	1 1 0 0 1 1 0 1
2	1 1 0 0 0 0 1 0
3	0 0 1 1 1 0 1 1

Дискретность

- Наименьшим элементом памяти является бит
- В одном бите памяти может храниться один бит информации

Адресуемость

- Байт памяти - наименьшая адресуемая часть внутренней памяти (1 байт = 8 бит)
- Все байты пронумерованы, начиная с 0
- Номер байта - адрес байта памяти
- Процессор обращается к памяти по адресам

Представление целых чисел

Положительные числа

1) Перевести число N в двоичную систему счисления [N_2]
2) Полученный результат [N_2] дополнить слева незначащими нулями до размеров машинного слова

1607_{10} 11001000111_1

0000 0110 0100 0111 — 16-разрядная ячейка памяти

$N_{16} = 0647$

$0111\ 1111\ 1111\ 1111_2 \rightarrow 7FFF_{16}$
$2^{15} - 1 = 32767_{10}$ (наибольшее число)

Диапазон значений:
$-2k^{-1} \le N \le 2k^{-1} - 1$
k - разрядность машинного слова

Отрицательные числа

1) $|-N| \rightarrow N_2 \rightarrow 00..N_2$ (положительное число)
2) Обратный код ($0 \rightarrow 1$, $1 \rightarrow 0$)
3) Обратный код +1 → результат

1) $|-1607_{10}| \rightarrow$ 0000 0110 0100 0111
2) Обратный код 1111 1001 1011
3) Прибавить единицу 1000
   результат: 1

$N_{16} = F9B9$ **(дополнительный код)**

Проверка N + (-N) = 0

```
  0000 0110 0100 0111   1607
  1111 1001 1011 1001  -1607
1 0000 0000 0000 0000      0
переполнение
```

$1000\ 0000\ 0000\ 0000 = -2^{15} = -32768_{10}$
(наименьшее число)

Развернутая форма записи числа

$$A_q \rightarrow a_{n-1} \times q^{n-1} + \ldots + a_1 \times q^1 + a_0 \times q^0 + a_1 \times q^{-1} + \ldots + a_{-m} \times q^{-m}$$

где **q** - основание системы счисления (количество используемых цифр)
A_q - число в системе счисления с основанием q
a - цифры многоразрядного числа A_q
n (m) - количество целых (дробных) разрядов числа A_q

Пример: $239{,}45_{10}$

$200 + 30 + 9 + 0{,}4 + 0{,}05$

$2 \times 100 + 3 \times 10 + 9 \times 1 + 4 \times 0{,}1 + 5 \times 0{,}01$

$2 \times 10^2 + 3 \times 10^1 + 9 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$

$a_2\ a_1\ a_0\ a_{-1}\ a_{-2} \rightarrow a_2 \times 10^2 + ф_1 \times 10^1 + a_0 \times 10^0 + a_{-1} \times 10^{-1} + a_{-2} \times 10^{-2}$

Таблица эквивалентности чисел в разных системах счисления

A_{10}	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

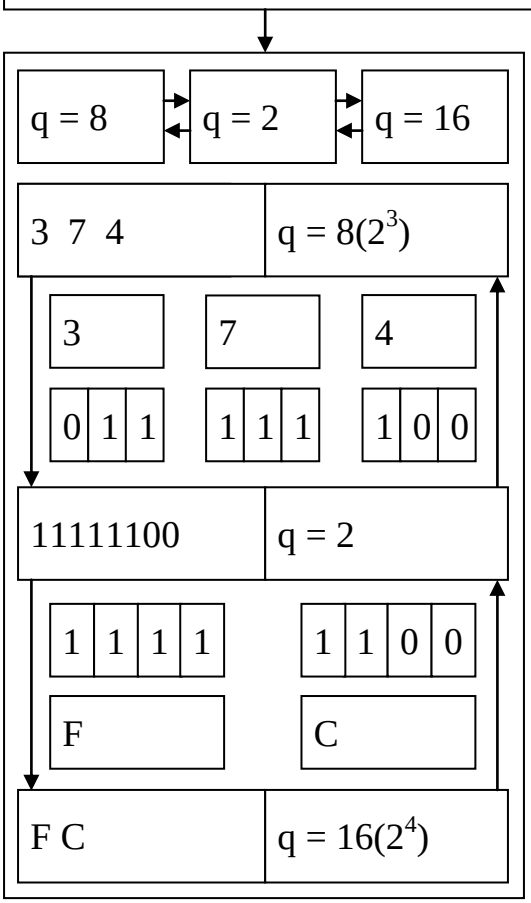
A_2	000	001	010	011	100	101	110	111								
A_8	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17

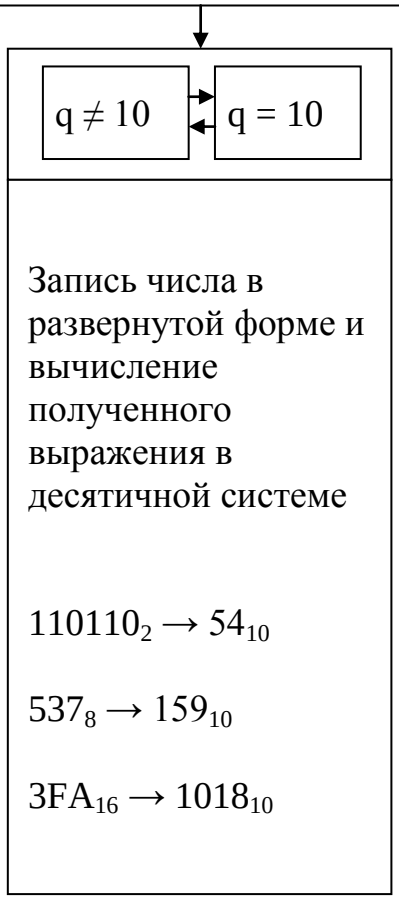
A_2	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
A_{16}	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

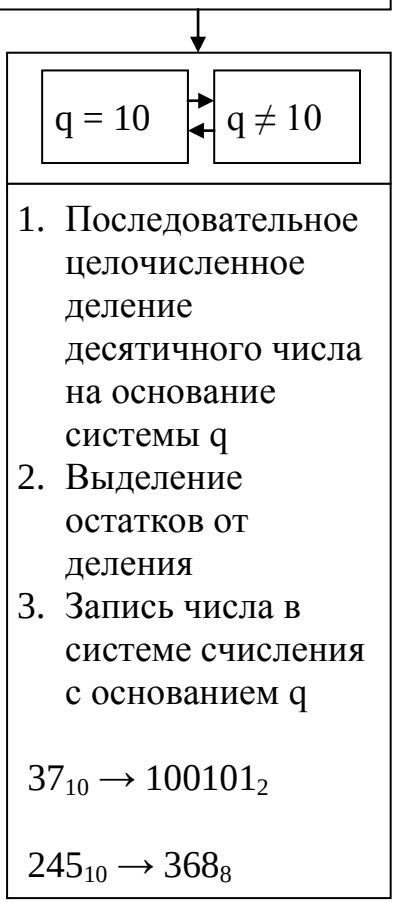
$A_2 \rightarrow A_8$ → 11111100 → 011 | 111 | 100 → 3 | 7 | 4 → 374 → 3 | 7 | 4 → 011 | 111 | 100

$A_2 \rightarrow A_{16}$ → 11111100 → 1111 | 1100 → F | C → FC → F | C → 1111 | 1100

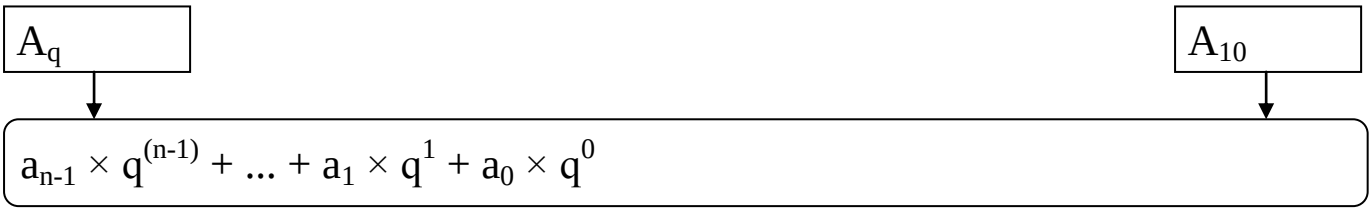
Перевод целых чисел из одной системы счисления в другую







Перевод чисел в десятичную систему счисления
(метод: использование развернутой формы записи числа)



$A_2 = 110110$	q = 2 (двоичное число)	n = 6
$1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 54$		

$A_8 = 237$	q = 8 (восьмеричное число)	n = 3
$2 \times 8^2 + 3 \times 8^1 + 7 \times 8^0 = 2 \times 64 + 3 \times 8 = 128 + 24 + 7 = 159$		

$A_2 = 3FA$	q = 16 (шестнадцатеричное число)	n = 3
$3 \times 16^2 + 15 \times 16^1 + 10 \times 16^0 = 3 \times 256 + 15 \times 16 + 10 = 1018$		