Учреждение образования
«Брестский государственный технический университет»

Факультет электронно-информационных систем
Кафедра «Интеллектуальные информационные технологии»

СОГЛАСОВАНО
Заведующий кафедрой
_____ В. А. Головко
«18» 12 2023 г.

СОГЛАСОВАНО
Декан факультета
_____ А. Н. Парфиевич
«18» 12 2023 г.

ЭЛЕКТРОННЫЙ УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

«КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ»

(название дисциплины)

для специальности (направления специальности):

1-40 01 01 Программное обеспечение информационных технологий

Составитель: Савицкий Юрий Викторович, к.т.н., доцент

Рассмотрено и утверждено на заседании Научно-методического совета университета 21.12.2023 г., протокол № 2.

рег. № УМК 23/24-04

Пояснительная записка

Дисциплина «Компьютерные системы и сети» является одной из базовых в процессе подготовки студентов по специальности 1-40 01 01 Программное обеспечение информационных технологий; относится к государственному компоненту; в значительной степени носит системный характер, формируя основу для освоения ряда последующих курсов, организации дипломного проектирования; обеспечивает выработку систематизированных знаний в области по архитектуре, логической организации, принципам построения и функционирования современных ЭВМ и сетей.

Цель преподавания учебной дисциплины:

- изучение основ организации, принципов построения и функционирования современных вычислительных комплексов, систем и сетей различного назначения; методам анализа их характеристик. Формирование у студентов систематизированного представления о принципах построения систем параллельной обработки информации. Получение практических навыков машинно-ориентированного и сетевого программирования.

Задачи учебной дисциплины:

- приобретение знаний по организации архитектур современных компьютерных систем и принципам их функционирования;
- изучение принципов построения современных компьютерных сетей, сетевых протоколов, сетевого программного обеспечения;
- овладение методами машинно-ориентированного и сетевого программирования.

Электронный учебно-методический комплекс (ЭУМК) объединяет структурные элементы научно-методического обеспечения образования и представляет собой сборник материалов теоретического и практического характера для организации работы студентов специальности 1-40 01 01 Программное обеспечение информационных технологий дневной формы получения образования по изучению дисциплины «Компьютерные системы и сети».

ЭУМК разработан на основании Положения об учебно-методическом комплексе на уровне высшего образования, утвержденного Постановлением Министерства образования Республики Беларусь от 8 ноября 2022 г. № 427.

ЭУМК разработан в полном соответствии с учебной программой Учреждения образования «Брестский государственный технический университет», утвержденной 29.06.2022 г., регистрационный № УД-221-171/уч. по учебной дисциплине государственного компонента «Компьютерные системы и сети».

Цели ЭУМК:

- обеспечение качественного методического сопровождения процесса обучения;
- организация эффективной самостоятельной работы студентов.

Содержание и объем ЭУМК полностью соответствуют образовательным стандартам высшего образования специальности 1-40 01 01 Программное обеспечение информационных технологий, а также учебно-программной документации образовательных программ высшего образования. Материал представлен на требуемом методическом уровне и адаптирован к современным образовательным технологиям.

Структура электронного учебно-методического комплекса по дисциплине «Компьютерные системы и сети»

Теоретический раздел ЭУМК содержит материалы для теоретического изучения учебной дисциплины и представлен конспектом лекций.

Практический раздел ЭУМК содержит материалы: для проведения лабораторных и практических учебных занятий в виде методических указаний для выполнения лабораторных и практических работ; для организации курсового проектирования в виде методических указаний для реализации всех этапов выполнения курсового проекта.

Раздел контроля знаний ЭУМК содержит материалы для итоговой аттестации (экзаменационные вопросы, практические задачи), позволяющие определить соответствие результатов учебной деятельности обучающихся требованиям образовательных стандартов высшего образования и учебно-программной документации образовательных программ высшего образования.

Вспомогательный раздел включает учебную программу учреждения высшего образования по учебной дисциплине «Компьютерные системы и сети».

Рекомендации по организации работы с ЭУМК:

- лекции проводятся с использованием представленных в ЭУМК учебных теоретических материалов; при подготовке к экзамену, лабораторным занятиям студенты могут использовать конспект лекций и набор практических задач;
- лабораторные занятия проводятся в компьютерной лаборатории с использованием: компьютерной сети рабочих станций; представленных в ЭУМК методических указаний; инструментальных программных средств ЭВМ – C++ Visual Studio, TASM, MASM, Wireshark, Cisco Packet Tracer, MS Word, MS Excel;
- практические занятия проводятся в условиях учебной аудитории с использованием представленных в ЭУМК методических указаний;
- курсовое проектирование осуществляется на базе представленных в ЭУМК методических указаний, с использованием ЭВМ и компьютерной сети, обладающих соответствующими характеристиками и конфигурацией для реализации процесса разработки сетевых и параллельных приложений обработки информации;
- экзаменационные материалы приведены в разделе контроля знаний.

ОГЛАВЛЕНИЕ

1 ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

1.1 КОНСПЕКТ ЛЕКЦИЙ

ОГЛАВЛЕНИЕ

I Организация компьютерных систем

1 Понятие архитектуры ЭВМ

Однозначно определить понятие архитектуры ЭВМ довольно трудно, потому что при желании в него можно включить все, что связано с компьютерами вообще и какой-то конкретной моделью компьютера в частности. Попытаемся все же его формализовать.

Архитектура ЭВМ — это абстрактное представление ЭВМ, которое отражает ее структурную, схемотехническую и логическую организацию. Понятие архитектуры ЭВМ является комплексным, в него входят:

- структурная схема ЭВМ;
- организация и разрядность интерфейсов ЭВМ;
-набор и доступность регистров;
- организация и способы адресации памяти;
- способы представления и форматы данных ЭВМ;
- набор и форматы машинных команд ЭВМ;
- правила обработки нештатных ситуаций (прерываний).

Таким образом, описание архитектуры включает в себя практически всю необходимую для программиста информацию о компьютере. Понятие архитектуры ЭВМ — иерархическое. Допустимо вести речь как об архитектуре компьютера в целом, так и об архитектуре отдельных его составляющих, например, архитектуре процессора или архитектуре подсистемы ввода-вывода.

Рассмотрим структурную схему типичного современного персонального компьютера. Она не претендует на безусловную точность и имеет целью лишь показать назначение, взаимосвязь и типовой состав его элементов.

На рисунке 1.1 показана функциональная схема системного блока компьютера на базе процессоров семейства Intel. На схеме представлены: центральный процессор, оперативная память, внешние устройства.

Все компоненты соединены между собой через системную шину. Системная шина имеет дополнительную шину — шину расширения. В компьютерах на базе Pentium в качестве такой шины используется шина PCI (Peripheral Component Interface), к которой подсоединяются внешние устройства, а также шины более ранних стандартов, например, ISA (Industry Standard Architecture).

На рисунке показана общая схема главного элемента компьютера — процессора. Основу процессора составляют блок микропрограммного управления, исполнительное устройство, обозначенное как «конвейер», и регистры. Остальные компоненты процессора в целом выполняют вспомогательные функции.

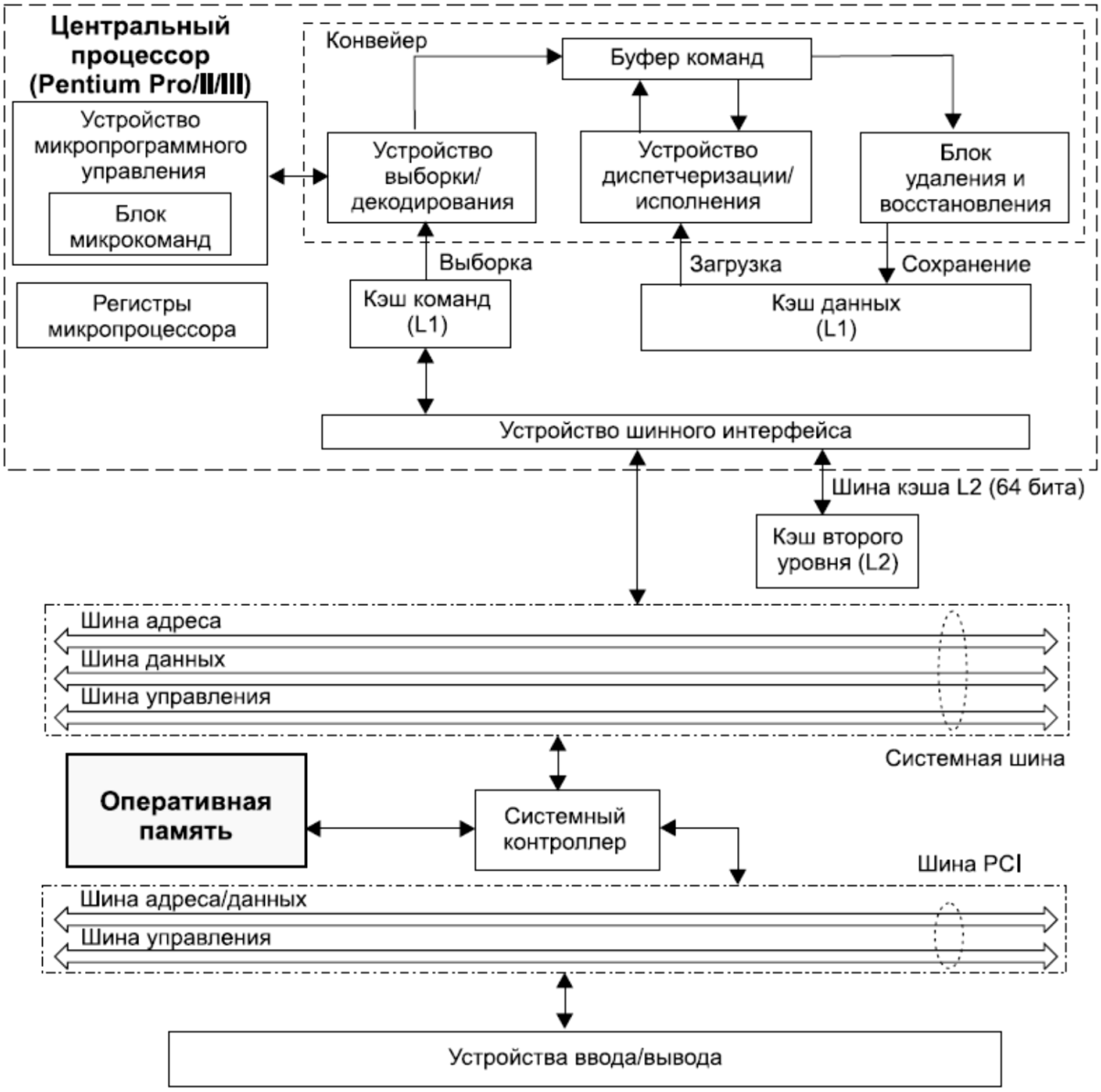


Рисунок 1.1 – Функциональная схема системного блока компьютера на базе процессоров семейства Intel

Все современные компьютеры обладают некоторыми общими и индивидуальными архитектурными свойствами. Индивидуальные свойства присущи только конкретной модели компьютера. Общие архитектурные свойства, наоборот, присущи некоторой, часто довольно большой группе компьютеров. На сегодняшний день общие архитектурные свойства большинства современных компьютеров подпадают под понятие *фон-Неймановской архитектуры*. Так названа архитектура по имени ученого фон Неймана. Когда фон Нейман начал заниматься компьютерами, программирование последних осуществлялось способом коммутирования. В первых ЭВМ для генерации нужных сигналов необходимо было с помощью

переключателей выполнить ручное программирование всех логических схем. В первых машинах использовали десятичную логику, при которой каждый разряд представлялся десятичной цифрой и моделировался 10 электронными лампами. В зависимости от нужной цифры одна лампа включалась, остальные девять оставались выключенными. Фон Нейман предложил схему ЭВМ с программой в памяти и двоичной логикой вместо десятичной. Логически машину фон Неймана составляли пять блоков (рисунок 1.2): оперативная память, арифметико-логическое устройство (АЛУ) с аккумулятором, блок управления, устройства ввода и вывода. Особо следует выделить роль аккумулятора. Физически он представляет собой регистр АЛУ. Для процессоров Intel, в которых большинство команд — двухоперандные, его роль не столь очевидна. Но существовали и существуют процессорные среды с одноопернадными машинными командами. В них наличие аккумулятора играет ключевую роль, так как большинство команд используют его содержимое в качестве либо второго, либо единственного операнда команды.

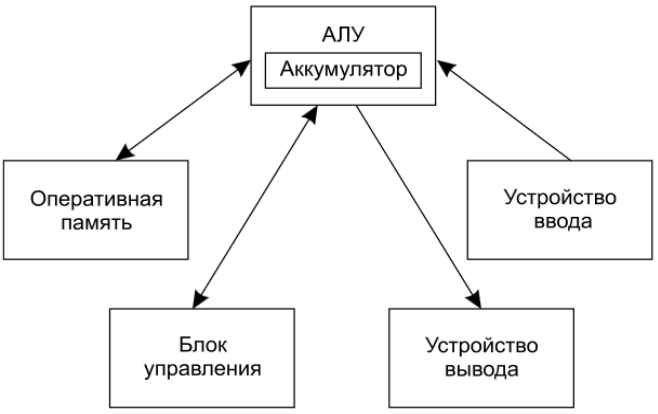


Рисунок 1.2 – Схема машины фон Неймана

Ниже описаны свойства и принципы работы машины фон Неймана.

Линейное пространство памяти. Для оперативного хранения информации компьютер имеет совокупность ячеек с последовательной нумерацией (адресами) 0, 1, 2, … Данная совокупность ячеек называется оперативной памятью.

Принцип хранимой программы. Согласно этому принципу, код программы и ее данные находятся в одном и том же адресном пространстве оперативной памяти.

Принцип микропрограммирования. Суть этого принципа заключается в том, что машинный язык еще не является той конечной субстанцией, которая физически приводит в действие процессы в машине. В состав процессора входит устройство микропрограммного управления, поддерживающее набор действий-сигналов, которые нужно сгенерировать для физического выполнения каждой машинной команды.

Последовательное выполнение программ. Процессор выбирает из памяти команды строго последовательно. Для изменения прямолинейного хода выполнения программы или осуществления ветвления необходимо использовать специальные команды. Они называются командами условного и безусловного переходов.

Отсутствие разницы между данными и командами в памяти. С точки зрения процессора, нет принципиальной разницы между данными и командами. Данные и машинные команды находятся в одном пространстве памяти в виде последовательности нулей и единиц. Это свойство связано с предыдущим. Процессор, поочередно обрабатывая некоторые ячейки памяти, всегда пытается трактовать содержимое ячеек как коды машинных команд, а если это не так, то происходит аварийное завершение программы. Поэтому важно всегда четко разделять в программе пространства данных и команд.

Безразличие к назначению данных. Машине все равно, какую логическую нагрузку несут обрабатываемые ею данные.

В рамках данного курса будут рассмотрены вопросы организации и программирования процессоров фирмы Intel и Intel-совместимых процессоров других фирм. Поэтому в качестве примера индивидуальных архитектурных принципов компьютера, в силу иерархичности этого понятия, рассмотрим индивидуальные архитектурные принципы и свойства процессоров Intel.

Архитектура IA-32 как предмет изучения. Согласно материалам фирмы Intel, индивидуальные архитектурные свойства и принципы работы всех ее процессоров, начиная с i8086 и заканчивая Pentium IV, выстроены в рамках единой архитектуры, которая позже получила название IA-32 (32-bit Intel Architecture). Архитектура IA-32 крайне удобна для изучения принципов организации компьютерных систем, поскольку, с одной стороны, дает представление об эволюции процессоров семейства Intel и ЭВМ на их базе, а с другой стороны, предоставляет слушателю информационный контент о развитии и особенностях более сложных архитектур, например IA-64, x86-64 и др.

2 Организация, классификация, характеристики процессоров

Микропроцессор — это универсальное программно-управляемое устройство цифровой электроники, осуществляющее прием, обработку и выдачу цифровых двоичных данных в соответствии с программой, хранящейся в запоминающем устройстве, и реализованное в виде одной или нескольких интегральных схем.

Основными характеристиками процессора являются:

- технологический процесс;
- тактовая частота (внутренняя и внешняя);
- производительность;

- потребляемая (рассеиваемая) мощность;
- тип корпуса (разъема), количество выводов;
- архитектура.

Технологический процесс определяется размером электронного элемента, создаваемого на кристалле. Обычные значения этой величины — 130, 90, 65, 45, 32 нм. Чем меньше размер элемента, тем соответственно, большее их число (число транзисторов) можно разместить на кристалле одной площади. Основным электронным элементом интегральной схемы является транзистор. Поэтому характеристикой процессора является именно число транзисторов на кристалле.

Тактовая частота и производительность процессора. Тактовая частота определяет быстродействие процессора. Для процессора различают внутреннюю (собственную) тактовую частоту процессора (с таким быстродействием могут выполняться внутренние простейшие операции) и внешнюю (определяет скорость передачи данных по внешней шине).

Для оценки производительности используются различные единицы и различные тесты. Наиболее распространенными основными единицами являются IPS (Instructions Per Second) и FLOPS (Floating Point Instructions Per Second), кратными —MIPS, MFLOPS, GFLOPS (M — Mega, G — Giga). Кроме того, существуют различные виды производительности: пиковая и реальная. Пиковая производительность — это теоретический максимум быстродействия процессора. Она определяется как максимально возможное число вычислительных операций, выполняемое в единицу времени всеми обрабатывающими устройствами процессора. Пиковая производительность связана с тактовой частотой процессора, разрядностью операндов, числом циклов (тактов) в команде. Например, Pentium за один такт может формировать один результат 64-разрядной операции с плавающей точкой или два результата 32-разрядных целочисленных операций. Таким образом, для Pentium 133 пиковая производительность равна 133 MFLOPS при выполнении вычислений с плавающей точкой и 266 MIPS при выполнении целочисленных 32-разрядных вычислений. Реальная производительность будет ниже, хотя и ненамного. SiSoftware Sandra 2005 показывает, например, такие цифры: 131 MFLOPS и 236 MIPS.

Реальная производительность ниже пиковой, теоретической или идеальной, по двум причинам:

- в определении пиковой производительности предполагается, что вся необходимая информация для выполнения программы, команды и данные, всегда доступна процессору. Однако в действительности это не так. Канал между процессором и памятью — это т. н. «бутылочное горлышко» (bottle neck) фон-неймановской архитектуры. Поэтому реальная производительность может быть увеличена путем совершенствования интерфейса процессора с памятью.

- не все параллельные блоки могут быть загружены в каждый момент времени. Для их полной загрузки необходимо, чтобы у исполняемой программы была необходимая степень внутреннего параллелизма, которую можно определить как долю команд, которые могут быть выполнены параллельно. Таким образом, создание параллельного кода — второй способ увеличения реальной производительности «параллельных» процессоров.

Тактовая частота и потребляемая мощность. Ранее было отмечено, что производительность компьютера можно повысить увеличением тактовой частоты. Однако, при этом увеличивается другая характеристика, связанная с тактовой частотой — потребляемая (рассеиваемая) мощность, которая пропорциональна квадрату напряжения питания и тактовой частоте. Рассеивание мощности – это процесс, в котором центральные процессоры потребляют электрическую энергию и рассеивают эту энергию в виде тепла за счет сопротивления в электронных схемах.

Основные типы корпуса процессоров:

- DIP (Dual In Line Package) — корпус с двухрядным расположением контактов;
- QFP (Quad Flatpack Package) — корпус с четырехрядным расположением выводов;
- PGA (Pin Grid Array) — корпус с решетчатой структурой штырьковых выводов, расположенных по нижней поверхности корпуса;
- BGA (Ball Grid Array) — корпус с решетчатой структурой шарообразных выводов, расположенных по нижней поверхности корпуса;
- картридж с однорядным расположением контактов (SECC, Single Edge Contact Cartridge).

Классификация процессоров по архитектуре. По совокупности различных архитектурных признаков выделяется множество типов процессоров:

- RISC (Reduced (Restricted) Instruction Set Computer) и CISC (Complex (Complete) Instruction Set Computer);
- Гарвардской и Принстонской архитектуры;
- со скрытым (implicit) параллелизмом (суперскалярные процессоры) и явным (explicit) параллелизмом (EPIC-процессоры);
- процессоры с фиксированной и модифицируемой системой команд.

RISC- и CISC-процессоры. RISC (Reduced (Restricted) Instruction Set Computer) — компьютер (процессор) с сокращенным набором команд. У RISC-процессоров система команд меньше, команды более простые и выполняются быстрее. Однако программа для RISC-процессоров длиннее. Примеры RISC-процессоров — Power PC компании IBM и SPARC компании Sun Microsystems.

CISC (Complex (Complete) Instruction Set Computer) — компьютер (процессор) с полным (сложным) набором команд. У CISC-процессоров система команд больше, команды более сложные и выполняются медленнее, но программа для них короче. Пример CISC-процессоров — семейство x86.

В принципе, можно было бы сделать процессор с еще более сложной системой команд, соответствующей, например, языку высокого уровня. Это было бы аппаратной реализацией языка высокого уровня. Но сейчас на практике так не делается, и языки высокого уровня "реализуются" программно, с помощью компиляторов и интерпретаторов. А сложные команды CISC-процессоров отображаются на микропрограммы, состоящие из микрокоманд.

Множество современных процессоров сочетает в себе свойства RISC- и CISC-процессоров, поскольку в процессе развития микропроцессорной техники в CISC-процессорах все больше появлялось элементов RISC, а в RISC, соответственно — CISC. Иллюстрацией этого являются процессоры Intel Pentium и IBM Power PC.

Простые команды RISC-процессоров позволяют эффективно использовать для своего исполнения конвейер. Дополнительный уровень детализации команд в CISC-процессорах, микрокоманды, позволяет использовать конвейер и в этом типе процессоров. Микрооперации и конвейер — это и есть элементы RISC в CISC-процессорах.

В RISC-процессорах оптимизация исполняемого кода в основном выполняется компилятором. В CISC-процессорах с элементами RISC это сделать невозможно по причине того, что машинные команды таких процессоров далее в процессе выполнения распадаются на еще более мелкие функциональные единицы — микрооперации. Поэтому в них оптимизация осуществляется в основном во время исполнения программы самим процессором путем переупорядочивания микроопераций и распределения их по параллельным блокам. Несмотря на то, что эти процессоры с точки зрения программиста и с точки зрения проектировщика различаются очень сильно, это можно считать их преимуществом, поскольку программа, скомпилированная для конкретной модели процессора, будет эффективно выполняться и на других моделях. Поэтому и в RISC-процессорах появляются оптимизирующие элементы.

Гарвардская и Принстонская архитектура. Гарвардская архитектура характеризуется раздельной памятью команд и данных. Она была реализована в Гарвардском университете в компьютере Mark I.

Основным признаком Принстонской архитектуры является объединенная (смешанная) память команд и данных (что предоставляет широкие возможности по модификации команд). Она ассоциируется с архитектурой фон Неймана, который был профессором математики в Принстонском Институте сложных (специальных) исследований.

Многие современные процессоры сочетают также и эти типы

архитектур. Например, процессоры x86 используют смешанные основную память и кэш второго уровня, но раздельный кэш первого уровня.

Суперскалярные и EPIC-процессоры. Последние два типа (со скрытым и явным параллелизмом) выделяются по признаку отражения в системе команд внутреннего параллелизма процессора.

Процессоры со скрытым (implicit) параллелизмом называются супескалярными. Суперскалярный процессор содержит не менее двух конвейеров. Конвейеры работают параллельно. Распараллеливание (распределение команд по конвейерам) происходит на аппаратном уровне. Команды суперскалярных процессоров не содержат в явном виде информации, касающейся параллельной обработки внутри процессора. Соответственно, компилятор не может оптимизировать программу для параллельной обработки. Пример суперскалярного процессора — Pentium.

Процессоры с явным (explicit) параллелизмом реализуют технологию вычислений под названием EPIC (Explicitly Parallel Instruction Computing). Распараллеливание происходит уже на уровне компилятора, поскольку команды в явном виде содержат информацию, касающуюся параллельной обработки внутри процессора. Это называется параллелизмом на уровне команд (ILP, Instruction-Level Parallelism). Примером EPIC-процессора является Intel Itanium.

Подклассом EPIC-процессоров являются процессоры с длинным командным словом (VLIW, Very Long Instruction Word). Команды VLIW-процессора имеют специальные поля, содержащие информацию для каждого из параллельных обрабатывающих устройств процессора, предписывающую им определенные действия.

Процессоры с фиксированной и модифицируемой системой команд. По возможности изменения системы команд выделяют два типа процессоров:

- процессор с фиксированной системой команд (с произвольной логикой);
- процессор с модифицируемой системой команд (с возможностью микропрограммирования).

Большинство процессоров имеет фиксированную систему команд. Микропрограммируемые процессоры оснащаются управляющим запоминающим устройством, в котором хранится микропрограмма — встроенные программы, определяющие набор выполняемых команд. Микропрограммное управление дает гибкость в ущерб быстродействию, а также позволяет легче скорректировать ошибки в выполнении команд.

Многие современные процессоры сочетают и эти крайние архитектурные признаки. Например, Pentium — процессор с фиксированной системой команд, но в нем имеется также и ПЗУ микрокоманд. При этом благодаря использованию конвейера и других технологических решений, указанный недостаток микропрограммного управления снимается. А начиная

с процессора Pentium 4 или микроархитектуры NetBurst, кэш-память команд первого уровня хранит уже не команды, а микрокоманды.

Специализированные процессоры и их функциональные характеристики. По типу обрабатываемых данных, а также совокупности других архитектурных признаков выделяются универсальные и специализированные процессоры. Специализированные процессоры часто являются основным элементом соответствующей подсистемы ЭВМ, но могут выступать и в качестве центрального процессора, например, сетевой процессор является центральным процессором в маршрутизаторах. Все специализированные процессоры также имеют свою систему команд, выполняют программы, только команды их ориентированы на определенную область.

Основные типы специализированных процессоров:

- математический процессор (устройство с плавающей точкой — Floating Point Unit, FPU);
- физический процессор (Physics Processing Unit, PPU);
- сигнальный процессор (Digital Signal Processor, DSP);
- медийный процессор;
- графический процессор (Graphics Processing Unit, GPU);
- сетевой процессор (Network Processor, NP);
- криптографический процессор.

Математический процессор выполняет операции над числами с плавающей точкой. Раньше они выпускались в виде отдельных устройств, а сейчас входят в состав большинства универсальных процессоров.

Физический процессор выполняет операции, связанные с различными физическими процессами: движением твердых тел и жидкостей, столкновением частиц и т. п. Первый физический процессор PhysX компании AGEIA Technology (www.ageia.com) был выпущен в 2005 году.

Сигнальные процессоры предназначены для выполнения следующих функций цифровой обработки сигналов:

- фильтрации сигнала;
- свертки (смешения) двух сигналов;
- вычисления корреляционных функций сигналов;
- усиления, ограничения, трансформации сигналов;
- прямого и обратного преобразования Фурье сигнала.

Для их выполнения они на аппаратном уровне поддерживают базовые операции цифровой обработки сигналов, к которым относятся:

- умножение с накоплением;
- модульная адресная арифметика;
- нормировка результатов арифметических операций.

Для выполнения этих операций, в свою очередь, сигнальные процессоры имеют аппаратный умножитель, позволяющий выполнять умножение двух

чисел за один такт. Также в них реализована аппаратная поддержка программных циклов и кольцевых буферов.

Пример сигнального процессора — Zilog Z89373.

Сигналы существуют самых разных типов. Соответственно сигнальные процессоры могут иметь ту или иную специализацию. Наиболее известными типами сигнальных процессоров являются медийные, звуковые и графические процессоры.

Медийный процессор. Медийные процессоры предназначены для обработки мультимедийной информации: аудиосигналов, графики — а также для выполнения коммуникационных функций. Сюда относится кодирование-декодирование аудио и видео в формате MPEG, табличный синтез звука, ускорение трехмерной графики. Примером медийного процессора является процессор Philips TriMedia.

Звуковые процессоры также называются звуковыми контроллерами (пример — Yamaha YMF724F). В составе звуковой подсистемы персонального компьютера они используются вместе со звуковыми кодеками (пример — Analog Devices AD1888).

Графический процессор выполняет операции по обработке и построению изображений. Современные графические процессоры могут превосходить по вычислительной мощности центральные (универсальные) процессоры. Поэтому они часто используются и для общих научно-технических расчетов.

Сетевой процессор — это процессор, предназначенный для использования в сетевых устройствах. Его основной задачей является обработка пакетов (packet processing). В состав NP входит ядро, исполняющее программы, процессоры обработки пакетов (пакетные процессоры — packet processor, PP), аппаратные ускорители и интерфейсные блоки с ОЗУ и шинами (их может быть несколько для поддержки сетей различных типов).

Ядро выполняется на базе универсального процессора. На нем исполняется все ПО, в том числе ОС и средства разработки.

PP выполняет следующие функции:

1) получает с сетевого интерфейса пакеты, ячейки или кадры и записывает их в ОЗУ;

2) определяет порядок обработки пакетов;

3) обрабатывает пакеты — осуществляет определение типа пакета, проверку политики, маршрутизацию, модификацию, присвоение QoS (качества обслуживания);

4) управляет перенаправлением пакетов. На основании QoS пакет может быть задержан;

5) отправляет пакет.

Функция 3 в PP реализуется программно, остальные — аппаратно. PP в целом реализуется как RISC-процессор или как макроячейка DSP-процессора.

Примеры сетевых процессоров: Intel IPX1200, Intel IPX465.

Основу IPX1200 составляет ядро 32-разрядного RISC-процессора

StrongARM. Он содержит шесть пакетных RISC-процессоров, поддерживает до 256 Мбайт ОЗУ типа SDRAM, включает интерфейсы с системной шиной PCI и 64-разрядной шиной IX Bus, поддерживающей набор сетевых интерфейсов ATM, T1/E1, 10, 100 и 1024 Мбит/с Ethernet и др.

Каждый PP имеет управляющую память в 1 Кслово и поддерживает четыре потока. Таким образом, IPX1200 одновременно может обрабатывать до 24 пакетов. Его пропускная способность — 1,2 Гб/с или 2,4 млн. пакетов/с.

Как правило, NP используют ОСРВ типа OS-9.

Криптографический процессор выполняет функции защиты информации. Примеры: SLD 9630 TT компании Infineon, AT97SC3201 и AT97SC3202 Atmel (код 38L3S13), PC8374T National Semiconductor, ST19WP18 STMicroelectronics.

3 Запоминающие устройства ЭВМ

Внутренняя и внешняя память. Запоминающие устройства ЭВМ делятся на внутренние и внешние.

К типу *внутренних* запоминающих устройств относятся оперативное запоминающее устройство (ОЗУ — RAM, Random Access Memory) и постоянное запоминающее устройство (ПЗУ — ROM, Read Only Memory). Запоминающее устройство в архитектуре фон Неймана является именно внутренним запоминающим устройством.

К типу *внешних* запоминающих устройств относятся:

- накопитель на жестких магнитных дисках (НЖМД или HDD, Hard Disk Drive) или просто жесткий диск, в просторечье винчестер;
- накопитель на гибких магнитных дисках (НГМД или FDD, Floppy Disk Drive);
- привод компакт дисков (CD-ROM, DVD-ROM);
- USB Flash и т. п.

Внешние запоминающие устройства относятся к устройствам ввода-вывода и имеют соответствующий контроллер.

Классификация внутренней памяти. ОЗУ и ПЗУ. По разным признакам можно выделять различные типы памяти ЭВМ. Главный признак классификации — возможность изменения информации в памяти в составе самой ЭВМ. По этому признаку выделяются:

- оперативное запоминающее устройство (ОЗУ, RAM — Random Access Memory);
- постоянное запоминающее устройство (ПЗУ, ROM — Read Only Memory).

ОЗУ является энергозависимой, а ПЗУ — энергонезависимой памятью. При выключении питания компьютера информация в оперативной памяти теряется. Информация в ПЗУ не меняется в процессе работы компьютера и сохраняется при выключении питания. Однако это не значит, что ее нельзя

изменить в принципе. Многие ПЗУ допускают такую возможность. Однако изменение информации в большинстве типов ПЗУ осуществляется не в составе ЭВМ, а в специальном устройстве — программаторе ПЗУ. Процесс изменения информации в ПЗУ называется перепрограммированием.

ОЗУ ЭВМ содержит выполняющиеся в настоящий момент программы, в том числе операционную систему, загружаемые с устройств внешней памяти, и необходимые для их выполнения данные.

ПЗУ ЭВМ, специализированных для выполнения определенных задач, могут содержать управляющую программу, являющуюся по сути простой операционной системой.

ПЗУ персональных компьютеров содержит BIOS (Basic Input-Output System, базовую систему ввода-вывода). Поэтому оно называется ROM BIOS.

В составе персональных компьютеров есть также т. н. CMOS RAM — оперативная память, в которой хранятся настройки BIOS, которые можно поменять с помощью программы Setup BIOS. CMOS RAM питается от аккумулятора и поэтому сохраняет содержимое при выключении питания.

Обратно, содержимое ROM BIOS современных компьютеров может быть перезаписано новой версией BIOS, причем в составе самого компьютера. Такая возможность имеется у ПЗУ, изготовленных по технологии Flash. Никакого программатора для перепрограммирования Flash-ROM не требуется. Однако это делается, конечно, не в процессе обычной работы компьютера, а в особом режиме с помощью специальных программных средств.

Существуют и смешанные типы. Так, современные модули оперативной памяти DIMM SDRAM содержат в своем составе постоянную память SPD EEPROM (Serial Presence Detect), которая идентифицирует тип модуля, различные параметры организации, временные параметры.

Классификация ПЗУ такова:

1) однократно программируемые:

- масочные ПЗУ — программируемые в процессе изготовления путем наложения маски;

- электрически программируемые ПЗУ (ЭППЗУ) — программируемые после изготовления пережиганием перемычек электрическим путем в программаторе;

2) многократно программируемые (перепрограммируемые):

- электрически стираемое программируемое ПЗУ (ЭСППЗУ, EEPROM, electrically erasable programmable read-only memory);

- со стиранием ультрафиолетовым светом и электрической записью — УФППЗУ.

Классификация ОЗУ. Основные качественные признаки классификации и выделяемые по ним классы ОЗУ персональных компьютеров таковы:

1) архитектура: FPM (Fast Page Mode), EDO (Extended Data OUT) и др.;

2) запоминающий элемент: конденсатор — динамическая память (DRAM, Dynamic RAM), триггер — статическая память (SRAM, Static RAM);

3) конструктивное исполнение: SIMM (Single Inline Memory Module, модуль памяти с однорядным расположением контактов), DIMM (Dual Inline Memory Module, модуль памяти с двухрядным расположением контактов);

4) наличие синхронизации тактовыми импульсами: асинхронная память, синхронная память (SDRAM, Synchronous DRAM);

5) умножение скорости передачи по сравнению с частотой тактовых импульсов: RDRAM (Rambus DRAM, выпускается в конструктивах под названием RIMM (Rambus Inline Memory Module)) и DDR (Double Data Rate, память с удвоенной скоростью передачи данных, конструктив DIMM). Удвоенная скорость передачи данных памяти DDR достигается за счёт считывания команд и данных не только по фронту, как в SDRAM, но и по спаду тактового сигнала. Поколения DDR, DDR 2, DDR 3, DDR 4, DDR 5 отличаются друг об друга конструктивом, тактовой частотой, объёмом и улучшенной производительностью;

6) многоканальность: двухканальная и трехканальная DDR (DDR II и DDR III).

Основные количественные признаки классификации ОЗУ таковы:

- ёмкость;
- размер ячейки: бит, тетрада, байт;
- время доступа;
- для синхронной памяти — тактовая частота и количество тактов, требующихся для доступа.

Совершенствование архитектуры и технологии памяти. Способы увеличения производительности памяти. Канал между процессором и памятью является фактором, ограничивающим производительность компьютера, "бутылочным горлышком" (bottle neck) фон-неймановской архитектуры.

Существует два способа увеличения производительности любого тракта (даже обычной автомобильной дороги):

- увеличение ширины (разрядности шины);
- увеличение скорости (тактовой частоты).

Влияет на производительность также время доступа к памяти для чтения или записи.

Многоуровневая организация оперативной памяти. Важным способом увеличения производительности компьютера является многоуровневая организация оперативной памяти. Уровни выделяются по степени близости к ядру процессора, минимальное количество уровней — два:

- основная оперативная память — динамическая, менее дорогая и быстрая, более ёмкая;
- кэш-память (сверхоперативная память, Cache Memory) — статическая, более дорогая и быстрая, менее емкая.

Существуют различные виды кэш-памяти, выделяемые по разным признакам. По признаку конструктивного исполнения (физической

реализации) можно выделить:

- кэш-память в виде отдельных микросхем;
- кэш-память в одном корпусе с процессором, но на отдельном кристалле;
- кэш-память на одном кристалле с процессором.

Эти типы физической реализации, конечно, различаются архитектурными особенностями и другими характеристиками.

Кэш-память может быть разделена на уровни, которые также характеризуются увеличением производительности и уменьшением емкости по мере близости к ядру процессора.

В большинстве современных процессоров кэш-память находится на кристалле процессора и делится на два или три уровня, которые обозначаются L1, L2 и L3, соответственно.

4 Организация, классификация, характеристики интерфейсов ЭВМ

Интерфейс - совокупность линий и шин, сигналов, электронных схем и алгоритмов (протоколов), предназначенных для осуществления обмена информацией между устройствами.

Под стандартными интерфейсами. понимают интерфейсы, которые приняты и рекомендованы в качестве обязательных отраслевыми или государственными стандартами.

Интерфейсы характеризуются следующими параметрами:

- пропускной способностью интерфейса – количеством информации, которое может быть передано через интерфейс в единицу времени;
- максимальной частотой передачи информационных сигналов через интерфейс;
- информационной шириной интерфейса – числом бит или байт данных, передаваемых параллельно через интерфейс;
- максимально допустимым расстоянием между соединяемыми устройствами;
- динамическими параметрами интерфейса – временем передачи отдельного слова или блока данных с учётом продолжительности процедур подготовки и завершения передачи;
- общим числом проводов (линий) в интерфейсе.

Можно выделить следующие четыре классификационных признака интерфейсов:

- способ соединения компонентов системы (радиальный, магистральный, смешанный);
- способ передачи информации (параллельный, последовательный, параллельно-последовательный);
- принцип обмена информацией (асинхронный, синхронный);
- режим передачи информации (двусторонняя поочередная передача, односторонняя передача).

Радиальный интерфейс даёт возможность всем модулям работать независимо с центральным модулем (ЦМ). Магистральный интерфейс (общая шина) использует принцип разделения времени для связи между ЦМ и другими модулями. Он прост в реализации, но лимитирует скорость обмена.

Параллельные интерфейсы позволяют передавать одновременно определенное количество бит или байт информации по многопроводной линии. Последовательные интерфейсы служат для последовательной передачи по двухпроводной линии.

В случае синхронного интерфейса моменты выдачи информации передающим устройством и приёма её в другом устройстве должны синхронизироваться, для этого используют специальную линию синхронизации. При асинхронном интерфейсе передача осуществляется по принципу «запрос-ответ».

Классификация интерфейсов по назначению содержит следующие уровни сопряжений:

- системные интерфейсы предназначены для организации связей между центральным процессором, ОП и контроллерами ПУ, и между процессорами в многопроцессорных системах;
- локальные интерфейсы предназначены для организации связи с отдельными устройствами компьютера, и для соединения микросхем чипсета между собой;
- назначение интерфейсов периферийных устройств (малых интерфейсов) состоит в выполнении функций сопряжения контроллера (адаптера) с конкретным механизмом ПУ;
- межмашинные интерфейсы используются в вычислительных системах и сетях.

Шина PCI. PCI (Peripheral Component Interconnect) – это компьютерная шина ввода/вывода, предназначена для подключения периферийных устройств к системной плате персонального компьютера. Шина PCI поддерживает 32-х/64-х битный обмен данными.

Частота шины PCI 33 МГц или 66 МГц (новые спецификации шины могут работать на более высоких частотах: 100 МГц, 133 МГц, 266 МГц, 533 МГц).

Характерной особенностью интерфейса PCI есть использование для передачи данных общей 32/64-битной двунаправленной параллельной шины, к которой подключаются все PCI-устройства. Любое устройство на шине PCI может позиционироваться как master-устройство (шина децентрализована).

Спецификации шины PCI. Первая версия шины (PCI 2.0) имела тактовую частоту 33 МГц, количество контактов — 124 (данные — 32) и 188 (данные — 64), скорость — 126 Мбайт/с и 252 Мбайт/с.

PCI 2.1 работает на частоте 33-66 МГц. Отсюда максимальная скорость — 66 МГц х 8 байтов (64 разряда) ≈ 504 Мбайт/с. Позволяет подключать до

десяти внешних устройств. Еще одним свойством этой шины является автоматическое конфигурирование внешних устройств (plug-n-play).

PCI-X — расширение PCI по частоте: 100, 133, 266 и 533 МГц. Пропускная способность (пиковая) – 1024 Мбайт/с (PCI-X 1.0) и 4096 Мбайт/с (PCI-X 2.0).

PCI Express — это шина, которая используется для подключения разнообразных комплектующих к настольной ЭВМ. С ее помощью подключают видеокарты, сетевые карты, звуковые карты, SSD накопители, WiFi модули и другие подобные устройства. Разработку данной шины начала компания Intel в 2002 году. Сейчас разработку новых версий данной шины занимается некоммерческая организация PCI Special Interest Group.

На данный момент шина PCI Express полностью заменила такие устаревшие шины как AGP, PCI и PCI-X.

Разработана на основе шины PCI. Основные отличия между PCI Express и PCI лежат на физическом уровне. В то время как PCI использует общую шину, в PCI Express используется топология типа звезда. Каждое PCI Express устройство подключается к общему коммутатору отдельным соединением.

Программная модель PCI Express во многом повторяет модель PCI. Поэтому большинство существующих CI контроллеров могут быть легко доработаны для использования шины PCI Express.

Кроме этого, шина PCI Express поддерживает такие новые возможности как:

- горячее подключение устройств;
- гарантированная скорость обмена данными;
- управление потреблением энергии;
- контроль целостности передаваемой информации;

Как работает шина PCI Express

Для подключения устройств шина PCI Express использует двунаправленное последовательное соединение. При этом такое соединение может иметь одну (x1) или несколько (x2, x4, x8, x12, x16 и x32) отдельных линий. Чем больше таких линий используется, тем большую скорость передачи данных может обеспечить шина PCI Express.

Пропускная способность PCI Express зависит от количества линий и версии шины (таблица 1.1).

Интерфейс IDE. IDE (Integrated Drive Electronics) или (ATA (AT Attachment). Первоначально использовался для подключения винчестеров. Разъем — 40 контактов (данные — 16). Имеет возможность подключения до двух устройств.

Существует несколько разновидностей интерфейса IDE, совместимых снизу-вверх друг с другом.

Таблица 1.1 – Пропускная способность PCI Express

В одну/обе стороны в Гбит/с							
	Количество линий						
	x1	x2	x4	x8	x12	x16	x32
PCIe 1.0	2/4	4/8	8/16	16/32	24/48	32/64	64/128
PCIe 2.0	4/8	8/16	16/32	32/64	48/96	64/128	128/256
PCIe 3.0	8/16	16/32	32/64	64/128	96/192	128/256	256/512
PCIe 4.0	16/32	32/64	64/128	128/256	192/384	256/512	512/1024

Существует расширение спецификации IDE для поддержки иных типов накопителей с интерфейсом IDE (приводов CD-ROM, CD-R и DVD-ROM, накопителей LS-120 и ZIP, магнитооптики, стримеров и т. п.), которое называется ATAPI (ATA Packed Interface).

Интерфейс IDE. IDE (Integrated Drive Electronics) или (ATA (AT Attachment). Первоначально использовался для подключения винчестеров. Разъем — 40 контактов (данные — 16). Имеет возможность подключения до двух устройств.

Существует несколько разновидностей интерфейса IDE, совместимых снизу-вверх друг с другом. Существует расширение спецификации IDE для поддержки иных типов накопителей с интерфейсом IDE (приводов CD-ROM, CD-R и DVD-ROM, накопителей LS-120 и ZIP, магнитооптики, стримеров и т. п.), которое называется ATAPI (ATA Packed Interface).

Интерфейс IDE поддерживает два типа обмена данными:

- программируемый ввод-вывод (PIO, Programmed Input/Output);
- прямой доступ к памяти (DMA, Direct Memory Access).

При программируемом вводе-выводе обмен данными производится через регистры процессора под его непосредственным управлением. В режиме прямого доступа к памяти контроллер интерфейса IDE и контроллер прямого доступа к памяти, расположенный на материнской плате, пересылают данные между диском и оперативной памятью, не загружая центральный процессор.

Скорость обмена зависит от разновидности интерфейса и способа обмена данными:

- PIO Mode 3 — 11,1 Мбайт/с;
- PIO Mode 4 и Single Word DMA Mode 2 — 16,7 Мбайт/с;
- Multi Word DMA Mode 2 — 20 Мбайт/с;
- Ultra ATA (Ultra DMA, ATA-33, DMA-33) при использовании режима DMA Mode 3 — 33.3 Мбайт/с;
- Ultra ATA-66 — 66 Мбайт/с;
- Ultra ATA/100 — 100 Мбайт/с.

Интерфейс SCSI. SCSI (Small Computer System Interface) используется для подключения винчестеров, стримеров, сменных жестких и магнитооптических дисков, сканеров, CD-ROM и CD-R, DVD-ROM и т. п.

(до 7 или 15 устройств, помимо контроллера).

Разъемы интерфейса SCSI отличаются большим разнообразием. Их можно разделить на два типа: для подключения внешних и внутренних устройств. Количество контактов: 25, 50 или 68. Существует множество разновидностей интерфейса SCSI, отличающихся шириной шины данных — 8 или 16 бит (Wide) и частотой тактовых импульсов:

- SCSI-1 — 5 МГц, 5 МБайт/с;
- Fast SCSI (SCSI-2) — 10 МГц, 10 МБайт/с;
- Fast Wide SCSI — 10 МГц, 20 МБайт/с;
- Ultra SCSI — 20 МГц, 20 МБайт/с;
- Ultra Wide SCSI — 20 МГц, 40 МБайт/с;
- Ultra 2 SCSI — 40 МГц, 40 МБайт/с;
- Ultra2 Wide SCSI — 40 МГц, 80 МБайт/с;
- Ultra3 SCSI — 40 МГц, 160 МБайт/с;
- Ultra-320 SCSI — 80 МГц, 320 МБайт/с;
- Ultra-640 SCSI — 160 МГц, 640 МБайт/с.

SATA. SATA (Serial ATA) является развитием параллельного интерфейса ATA (IDE). Существует несколько версий SATA:

- SATA 1.x — до 1,5 Гбит/с;
- SATA 2.x — до 3 Гбит/с;
- SATA 3.x — до 6 Гбит/с;

Serial Attached SCSI (SAS). Интерфейс Serial Attached SCSI (SAS) разработан для замены параллельного интерфейса SCSI. Позволяет достичь более высокой пропускной способности, чем SCSI — до 12 Гбит/с.

Интерфейс USB. USB — Universal Serial Bus, универсальная последовательная шина. Для подключения периферийных устройств к шине USB используется четырёхпроводный кабель, при этом два провода в дифференциальном включении используются для приема и передачи данных, а два провода — для питания периферийного устройства.

К одному порту может быть подключено до 127 устройств, включая концентраторы, через цепочку концентраторов (хабов).

Режимы и производительность USB 1.0:

- Low-Speed — 1,5 Мбит/с;
- Full-Speed — 12 Мбит/с.

Режимы и производительность USB 2.0:

- Low-speed — 10-1500 Кбит/с (клавиатура, мышь, джойстик);
- Full-speed — 0,5—12 Мбит/с (аудио-, видеоустройства);
- Hi-speed — 25—480 Мбит/с (видеоустройства, устройства хранения информации).

5 Эволюция архитектуры x86

5.1 Возникновение и развитие процессоров семейства x86

В 1968 году Роберт Нойс (Robert Noyce), изобретатель кремниевой интегральной схемы, Гордон Мур (Gordon Moore), автор известного закона Мура, и Артур Рок (Arthur Rock), венчурный капиталист из Сан-Франциско, основали корпорацию Intel для производства компьютерных микросхем. В настоящее время Intel является крупнейшим мировым производителем процессоров.

В 1969 году японская компания Busicom обратилась к корпорации Intel с просьбой выпустить 12 несерийных микросхем для электронной вычислительной машины. Благодаря нестандартным инженерным решениям в рамках этого проекта в 1970 году появился первый процессор на одной микросхеме — i4004 на 2300 транзисторах. Компания Intel начала работу над 8-разрядной версией микросхемы, 8008, выпущенной в 1972 году. Таблица 1.2 отражает эволюцию ряда процессоров семейства Intel, начиная с моделей 4004 и 8008. Поскольку никто не ожидал большого спроса на микросхему 8008, она была выпущена достаточно ограниченным тиражом. Новая микросхема вызвала большой интерес, поэтому компания Intel начала разработку еще одного процессора, в котором предел в 16 Кбайт памяти (как у процессора 8008), навязываемый количеством внешних выводов микросхемы, был преодолен.

Так появился небольшой универсальный процессор 8080, выпущенный в 1974 году. Как и PDP-8, он произвел революцию на компьютерном рынке и сразу стал массовым продуктом. Разница лишь в масштабах: компания DEC продала тысячи PDP-8, а Intel — миллионы процессоров 8080.

В 1978 году появился процессор 8086, 16-разрядный процессор на одной микросхеме. Процессор 8086 был во многом похож на 8080, но не был полностью совместим с ним. Затем появился процессор 8088 с такой же архитектурой, как у 8086. Он исполнял те же программы, что и 8086, но вместо 16-разрядной шины у него была 8-разрядная, из-за чего процессор работал медленнее, но стоил дешевле, чем 80861. Когда компания IBM выбрала процессор 8088 для IBM PC, эта микросхема быстро превратилась в промышленный стандарт в области персональных компьютеров.

Ни 8088, ни 8086 не могли адресовать память объемом более 1 Мбайт. К началу 80-х годов это стало серьезной проблемой, поэтому компания Intel разработала модель 80286, совместимую с 8086. Основной набор команд остался в сущности таким же, как у процессоров 8086 и 8088, но организация памяти была несколько иной — и довольно неудобной из-за требования совместимости с предыдущими микросхемами и могла работать по-прежнему. Процессор 80286 использовался в IBM PC/AT и в моделях PS/2. Он, как и 8088, пользовался большим спросом (главным образом потому, что покупатели рассматривали его как более быстрый вариант модели 8088).

Следующим шагом был 32-разрядный процессор 80386, выпущенный в 1985 году. Как и 80286, он был более или менее совместим со всеми старыми версиями. Совместимость такого рода оказывалась благом для тех, кто пользовался старым программным обеспечением, и некоторым неудобством для тех, кто предпочитал современную архитектуру, не обремененную ошибками и технологиями прошлого.

Таблица 1.2 – Эволюция процессоров Intel

Микросхема	**Дата выпуска**	**Тактовая частота, МГц**	**Количество транзисто-ров**	**Объем памяти**	**Примечание**
4004	4/1971	0,108	2 300	640 байт	Первый микропроцессор на микросхеме
8008	4/1972	0,08	3 500	16 Кбайт	Первый 8-разрядный микропроцессор
8080	4/1974	2	6 000	64 Кбайт	Первый многоцелевой процессор на микросхеме
8086	6/1978	5–10	29 000	1 Мбайт	Первый 16-разрядный процессор на микросхеме
8088	6/1979	5–8	29 000	1 Мбайт	Использовался в IBM PC
80286	2/1982	8–12	134 000	16 Мбайт	Появилась защита памяти
80386	10/1985	16–33	275 000	4 Гбайт	Первый 32-разрядный процессор
80486	4/1989	25–100	1 200 000	4 Гбайт	Кэш-память на 8 Кбайт
Pentium	3/1993	60–223	3 100 000	4 Гбайт	Два конвейера, у более поздних моделей — MMX
Pentium Pro	3/1995	150–200	5 500 000	4 Гбайт1	Два уровня кэш-памяти
Pentium II	5/1997	233–400	7 500 000	4 Гбайт	Pentium Pro плюс MMX
Pentium III	2/1999	650–1400	9 500 000	4 Гбайт	Появились SSE- команды, ускоряющие обработку трехмерной графики
Pentium 4	11/2000	1300–3800	42 000 000	4 Гбайт	Гиперпоточность, дополнительные SSE-команды
Core Duo	1/2006	1600–3200	152 млн	2 Гбайт	Два ядра на одной под-ложке
Core	7/2006	1200–3200	410 млн	64 Гбайт	64-разрядная 4-ядерная архитектура
Core i7	1/2011	1100–3300	1 160 млн	24 Гбайт	Интегрированный графический процессор

Через четыре года появился процессор 80486. Он работал быстрее, чем 80386, мог исполнять операции с плавающей точкой и имел кэш-память объемом 8 Кбайт. Кэш-память позволяет держать наиболее часто используемые слова внутри центрального процессора и избегать (медленных) обращений к основной памяти. Процессор 80486 содержал встроенную поддержку мультипроцессорного режима, что давало производителям возможность конструировать системы с несколькими процессорами.

Следующее поколение компьютеров (1993 год) получило название Pentium (от греческого слова πεντε — пять). Благодаря суперскалярной архитектуре (с двумя конвейерами, известными как u и v) он мог выполнять две машинные инструкции за один такт. К внутреннему кэшу команд добавлен внутренний 8-килобайтный кэш данных. Реализована технология предсказания переходов (branch prediction). Для увеличения пропускной способности при обработке данных процессор Pentium поддерживает внутренние пути шириной 128 и 256 битов, внешняя шина данных увеличена до 64 битов. Добавлены средства для построения многопроцессорных систем, в частности расширенный программируемый контроллер прерываний (Advanced Programmable Interrupt Controller, APIC), дополнительные выводы и специальный режим дуальной обработки (dual processing), ориентированный на построение двухпроцессорных систем.

Впоследствии в линейку Pentium были введены дополнительные команды, известные под общим названием MMX (MultiMedia eXtension — мультимедийное расширение). Они были предназначены для ускорения вычислительных операций, связанных с обработкой звуковых и видеоданных, что позволило отказаться от специальных мультимедийных сопроцессоров.

Когда появилось следующее поколение компьютеров, те, кто рассчитывал на название Sexium (sex по латыни — шесть), были разочарованы. Название Pentium стало так хорошо известно, что его решили оставить, и новую микросхему назвали Pentium Pro. Несмотря на столь незначительное изменение названия, этот процессор очень сильно отличался от предыдущего. У него была совершенно другая внутренняя организация и он мог исполнять до пяти команд одновременно.

Еще одно нововведение у Pentium Pro — двухуровневая кэш-память. Процессор содержал 8 Кбайт памяти для часто используемых команд и еще 8 Кбайт для часто используемых данных. В корпусе Pentium Pro рядом с процессором (но не на самой микросхеме) находилась другая кэш-память объемом в 256 Кбайт.

Большой объем кэш-памяти в Pentium Pro отчасти компенсировался отсутствием MMX-команд (первоначально Intel не удалось спроектировать микросхему адекватного размера, отвечавшую критерию рентабельности). Когда технологическая база позволила совместить в рамках одной микросхемы набор MMX-команд и большой кэш, новая модель получила

название Pentium II. Через некоторое время для улучшенной передачи трехмерной графики в процессор были введены дополнительные мультимедийные команды под названием SSE (Streaming SIMD Extensions — потоковые SIMD-расширения) — в результате появился процессор Pentium III. Правда, согласно внутренней номенклатуре компании это все тот же Pentium II.

Следующая модель Pentium получила новую внутреннюю архитектуру. Одновременно было решено перейти с римских цифр в обозначениях моделей на арабские. Так появился процессор Pentium 4. По традиции он превосходил все предыдущие модели по быстродействию. В версии с тактовой частотой 3,06 ГГц была реализована новая функция — гиперпоточность (hyper threading). Она позволяет программам разделять задачи на два программных потока, которые обрабатываются процессором параллельно; следовательно, скорость исполнения повышается. Кроме того, для дальнейшего повышения скорости обработки звуковых и видеоданных был внедрен дополнительный набор SSE-команд.

В 2006 году фирма Intel сменила название бренда Pentium на Core и выпустила двухъядерную микросхему Core 2 duo. Когда возникла необходимость в более дешевой одноядерной версии микросхемы, фирма Intel стала продавать Core 2 duo с одним отключенным ядром, потому что небольшие дополнительные затраты на каждой производимой микросхеме обходились несравненно дешевле огромных затрат на проектирование и тестирование новой модели «с нуля». Серия Core продолжала развиваться; модели i3, i5 и i7 стали популярными решениями для низко-, средне- и высокопроизводительных компьютеров.

Помимо основной линейки процессоров, которую мы рассмотрели, Intel разрабатывает специальные варианты микросхем для отдельных сегментов рынка. В начале 1998 года компания запустила новую линейку под названием Celeron. По сути, это был дешевый вариант Pentium 2 с пониженной производительностью для низкопроизводительных компьютеров. В июне 1998 года компания Intel выпустила специальную версию Pentium 2 для верхнего сегмента рынка — Xeon. Эту модификацию снабдили кэш-памятью большего объема, ускоренной внутренней шиной и усовершенствованными средствами поддержки мультипроцессорного режима, однако по всем остальным параметрам она ничем не отличалась от Pentium 2, что дает нам полное право не рассматривать ее отдельно. Для процессоров Pentium III, как и более поздних микросхем, также была разработана версия Xeon. На более новых процессорах одной из особенностей Xeon является увеличенное количество ядер.

В 2003 году появилась микросхема Pentium M (где M — сокращение от «Mobile») для портативных компьютеров. Она задумывалась как составная часть новой архитектуры Centrino, которая должна была, во-первых, снизить энергопотребление, увеличив тем самым ресурс аккумулятора, во-вторых, обеспечить возможность производства более компактных и легких корпусов,

в-третьих, предоставить встроенную поддержку беспроводных сетевых соединений по стандарту IEEE 802.11 (WiFi). Pentium M потребляет меньше энергии и отличается большей компактностью по сравнению с Pentium 4; вероятно, эти две характеристики вскоре позволят ему (и его преемникам) вытеснить микроархитектуру Pentium 4 в будущих продуктах Intel.

Все микросхемы Intel обратно совместимы со всеми своими предшественниками вплоть до модели 8086. Другими словами, программы, написанные когда-то для 8086, исполняются на Pentium 4 без каких бы то ни было изменений. Обратная совместимость в течение длительного времени является одним из основных принципов проектирования в Intel — соблюдение этого принципа позволяет сохранить предыдущие инвестиции в программное обеспечение. Естественно, поскольку модель Pentium 4 на три порядка сложнее, чем 8086, ее возможности несопоставимо шире. Из-за постепенных расширений, которые проектировщикам процессоров приходилось внедрять для достижения этой цели, архитектура получилась не такой элегантной, как если бы разработчики Pentium 4 получили 42 миллиона транзисторов и начали бы все строить «с нуля».

5.2 Технические особенности поколений модели Core i7

Core i7 – топовые процессоры от Intel, занимающие флагманские и субфлагманские позиции. До появления i9 они были самыми мощными, уступая только серверным моделям Xeon. Модельный ряд производится более 10 лет и рассчитан на использование в мощных игровых и рабочих компьютерах. За все это время создано множество поколений этой модели CPU; в каждой линейке есть несколько подсерий, которые отличаются рабочими параметрами.

Условно эти чипы можно разделить на стоковые и продвинутые. Последние имеют собственную «экосистему» из соответствующих системных плат, чипсетов и сокетов. Они относятся к так называемой серии X. Также в маркировке используются следующие обозначения: K – разблокированный множитель и поддержка разгона; S – сниженное энергопотребление; T – очень сниженное; E – CPU для встраиваемых систем; C и R – чипы с графикой Iris. Фотография микросхемы i7 приводится на рисунке 1.3. На ней в действительности размещено восемь ядер, но во всех версиях, кроме Xeon, включены только шесть. Такой подход означает, что микросхему с одним или двумя дефектными ядрами все равно можно продать — достаточно отключить дефектное ядро(-а). Каждое ядро имеет собственные кэши 1 и 2 уровня, но также имеется общий кэш 3 уровня (L3), используемый всеми ядрами.

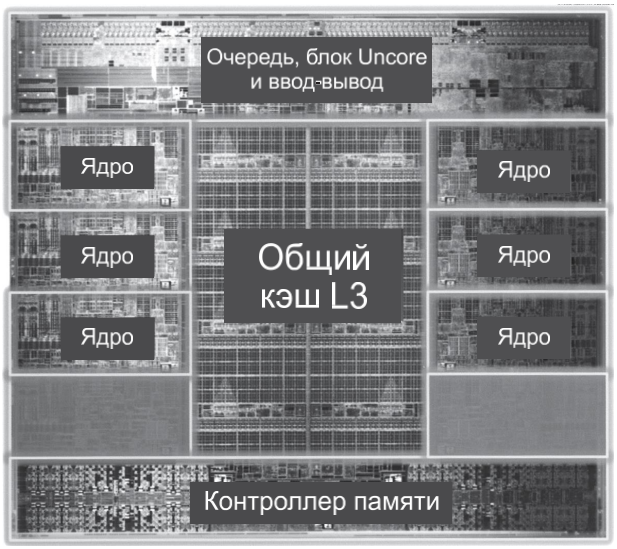


Рисунок 1.3 – Микросхема Intel Core i7-3960X. Подложка имеет размеры 21*21 мм и содержит 2,27 миллиарда транзисторов

1 поколение (2008). Чипы с модельными номерами 920, 930, 940, 950, 960, 965, 975 создавались по техпроцессу 45 нм. У всех CPU было по 4 ядра, которые работали в восемь потоков. Вершиной этого поколения стал CPU с архитектурой Gulftown. Такие CPU получили индексы 970, 980, 980X и 990X. Создавались они по 32 нм процессу и были шести ядерными. Поддерживали трехканальный режим памяти и подключались через сокет 1366.

2 поколение (2011). Архитектуру изменили на Snady Bridge и окончательно перешли на 32 нм техпроцесс. В базовой серии были выпущены процессоры 2600, 2600S, 2600K, 2700K – четырехъядерные, восьми потоковые, работали с одноканальной памятью и монтировались в новые 1155 сокеты. Логичным продолжением стала модель под платформу 2011, которая сменила устаревшую 1366. Это CPU с кодами 3820, 3930K, 3960X, 3970X. У младшей модели было 4 ядра, у старших 6. Новинкой стал четырехканальный контроллер для памяти DDR III.

3 поколение (2011-2012). Использовалась архитектура Ivy Bridge, доработанная версия предшественницы с техпроцессом 22 нм. В рамках линейки созданы чипы с индексами 3770, 3770S, 3770T, 3770K – четырехъядерные, с поддержкой двух каналов ДДР3.

4 поколение (2013-2014). В рамках серии X, выпущены модификации с кодовыми номерами 4820K, 4930K и 4960X. Техпроцесс 22 нм. Устанавливались в сокет 2001 и поддерживали 4 канала DDR3. Созданное большое число модификаций на архитектуре Haswell – 4765T, 4770, 4770K, 4770S, 4770T, 4770TE, 4771, 4785T, 4790, 4790T, 4790S, 4790K. Монтировались на платы с новым сокетом 1150 и имели встроенный графический чип HD 4600.

5 поколение (2014-2015). Техпроцесс остался прежним – 22 нм. В рамках серии X выпущены 5820K, 5930K и 5960X. Контроллер перевели на память ДДР4, поэтому использовалась платформа 2011 третьей версии. Массового производства процессоров этой серии не было. Производитель осваивал 14 нм техпроцесс на архитектуре Broadwell. Создано всего две модели: 5775C и 5775R – один и тот же чип с графическим ускорителем Iris Pro 6200. В серии X созданы модели 6800K, 6850K, 6900K и 6950X. Они работали с четырехканальной памятью ДДР 4 и ставились в слот 2011 третьей версии.

6 поколение (2015-2016). На 14 нм техпроцессе, производителем выпущено шестое поколение, представленное моделями 6700, 6700K, 6700T и 6700TE. Эти CPU имели по четыре ядра, встроенную видеокарту HD 530 и строились на архитектуре SkyLake. Двойной контроллер поддерживал ДДР3 и ДДР4. Монтировались на разъеме 1151. В топовой категории выпущено три модификации: 7800X, 7820X, 9800X. Устанавливались они в сокет 2066.

7 поколение (2017). Использована модернизированная архитектура Kaby Lake, которая выпускалась по техпроцессу 14 нм. Выпущены модели 7700, 7700T и 7700K. Совместимы с платами 1151. В X-серии выпущен всего один чип – 7740X, четырехъядерник для платформы 2066.

8 поколение (2017-2018). Ахитектура Coffee Lake. В модельный ряд включены 8700, 8700K и 8700T, которые имели по 6 ядер. Техпроцесс 14 нм. Сокет обновлен до 1151 второй версии, поддержку ДДР3 убрали. Ограниченным тиражом выпущен 8086K, приуроченный к 40-летию CPU Intel 8086.

9 поколение (2018-2019). Чипы, выпущенные в 2019 году, кардинальных нововведений не получили. Использована та же архитектура и тот же техпроцесс. Пока в последнем модельном ряду два процессора: 9700KF и 9700K. Работают в таких же платах, как CPU предыдущего поколения. Ядер у этих чипов уже по восемь.

10 поколение (2020). Серия известная как Comet Lake-S. В этих процессорах используется сокет LGA1200, который пришел на смену 1151-2 v2. В общей сложности планируется выпустить более трех десятков моделей CPU этого поколения(речь идет не только о десктопных вариантах). При их производстве использован улучшенный 14-нм тех. процесс, но от предшественников, Skylake-S, эти CPU в плане архитектуры почти не отличаются. Графический блок UHD Graphics и вовсе остался без изменений. Главное отличие от предшественников — более совершенные механизмы динамического разгона ядер.

11 поколение (2021). Эти процессоры создаются на архитектуре Cypress Cove. Разработка не новая — по сути, это оптимизированная Sunny Cove, на которой выпускаются мобильные процессоры Ice Lake. Планировалось, что будет использован 10 нм техпроцесс, однако для десктопных ПК схемы пришлось перенести на 14 нм. Такая трансформация не прошла бесследно. Флагманская модель лишилась пары ядер, и теперь их осталось восемь при шестнадцати потоках. Связано это с тем, что физически увеличился размер

кристалла, и при его стандартном размере не получилось вместить лишние ядра. По сравнению с предшественниками увеличены объем и производительность кеш-памяти. Произошло изменение ее иерархии: кеш L1 увеличен на 50%, объем кеша L2 удвоен. Кеш третьего уровня изменения не затронули. AGU и Store Data теперь могут обрабатывать две операции за цикл. Контроллер памяти официально поддерживает частоту ОЗУ до 3200 МГц. Как и прежде, используется сокет LGA1200.

12 поколение Core I7 (2021). Кодовое имя Alder Lake-S. Это первые процессоры с поддержкой памяти DDR5 и высокоскоростной шины PCI Express 5.0. Про пропускная способность шины в расчете на одну линию 7.87 Гбайт/с. Работают на сокете LGA1700 в связке с чипсетами 600-й серии. Эти процессоры вернули Intel лидерство по результатам тестов в игровых бенчмарках, однако расплачиваться приходится увеличенным энергопотреблением по сравнению с CPU конкурентов. Новая компоновка позволяет обрабатывать данные до 24 потоков.

13 поколение (2022-2023). Raptor Lake — кодовое имя семейства процессоров данного поколения. Чипы на этой микроархитектуре изготавливаются на технологическом процессе Intel 7 (10 нм). Получат поддержку разъёма LGA 1700 и, как и процессоры предыдущего поколения, смогут работать с памятью DDR5. В Intel Raptor Lake-S имеются до 16 энергоэффективных ядер Gracemontruen и до 8 высокопроизводительных Raptor Coveruen.

14 поколение (2023). Meteor Lake — кодовое имя семейства процессоров фирмы Intel 14-го поколения. В процессорах серии Meteor Lake компания Intel будет использовать кристаллы, произведённые сразу по трём разным технологическим процессам. Чипы на этой микроархитектуре изготавливаются на технологическом процессе Intel 4 (предполагается 2023 году выпуск гибридных процессоров Meteor Lake по технологии 3 нм). В конструкции многокристального модуля Intel используется технология упаковки Intel Foveros. Meteor Lake получат новый сокет LGA 1851.

5.3 Архитектура IA-32

Варианты микроархитектуры процессоров Intel. Понятие микроархитектуры впервые было определено Intel для процессоров семейства Pentium Pro. Его введение объяснялось необходимостью правильного позиционирования новых процессоров среди существующих. Внешняя программная модель (логическая) 32-разрядных процессоров изменялась только в сторону развития, в то время как их исполнительная (физическая) часть могла быть совершенно разной. Понятие микроархитектуры ориентировано на описание особенностей исполнительной части процессоров, то есть того, какими способами и какими средствами процессор выполняет обработку машинного кода (рисунок 1.4). В частности, день в рамках IA-32 существует две микроархитектуры процессоров Intel: P6 и

NetBurst.

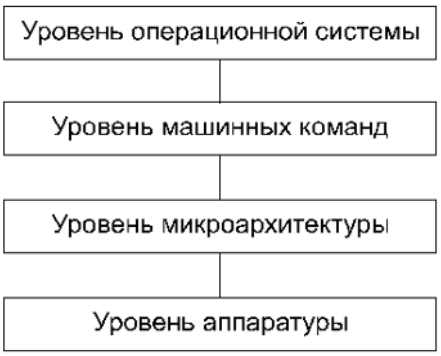


Рис. 1.4 – Представление компьютера в виде уровней

5.3.1 Микроархитектура P6

Микроархитектуру P6 поддерживают такие процессоры Intel, как Pentium Pro, Pentium II (Xeon), Celeron, Pentium III (Xeon). Эта микроархитектура является, по определению Intel, *трехходовой* (three-way) *суперскалярной конвейерной* архитектурой. Термин «трехходовая» означает поддержку технологий параллельного вычисления, позволяющих процессору одновременно (за один такт) обрабатывать до трех инструкций. Проблема оптимальной обработки потока машинных команд является ключевой при разработке любого процессора. Поэтому для большей ясности необходимо показать эту проблему в развитии. В компьютере фон-неймановской архитектуры существуют две основные стадии исполнения команды — выборка очередной команды из памяти и собственно ее исполнение. В первых процессорах Intel все блоки процессора работали последовательно, начиная с этапа выборки очередной команды из памяти и заканчивая этапом завершения ее обработки процессором. Напоминание об этом осталось в названии регистра IP/EIP — (Instruction Pointer — указатель инструкции). До появления процессоров Intel с конвейерной архитектурой данный регистр непосредственно указывал на очередную команду, подлежащую выполнению. Процессоры Intel относятся к группе CISC-процессоров, в которых для выполнения одной команды может требоваться от единиц до нескольких десятков процессорных тактов. При такой обработке команд увеличение производительности может быть достигнуто только повышением частоты генерации машинных тактов. Простое увеличение частоты работы процессора не имеет смысла, так как есть физически обусловленная верхняя граница, до которой ее можно поднимать. По этому пути разработчики Intel шли до процессора i80386 включительно. В ходе исполнения команды есть и другое узкое место — выборка команды из памяти. Это затратная по времени операция. Частичное решение проблемы было найдено еще на заре развития компьютерной техники в виде буфера упреждающей выборки. Развитием этой и реализацией других идей стал

конвейер — специальное устройство, существующее на уровне архитектуры исполнительной части компьютера. Благодаря конвейеру исполнение команды разбивается на несколько стадий, каждая из которых реализует некоторую элементарную операцию общего процесса обработки команды. Впервые для процессоров Intel конвейер был реализован в архитектуре процессора i80486. Конвейер i80486 имеет пять ступеней, которые соответствуют перечисленным далее стадиям обработки машинной команды.

1. Выборка команды из кэш-памяти или из оперативной памяти.
2. Декодирование команды.
3. Генерация адреса, в ходе которой определяются адреса операндов в памяти и выполняется выборка операндов.
4. Выполнение операции с помощью АЛУ.
5. Запись результата (место записи результата зависит от алгоритма работы конкретной машинной команды).

В чем преимущество такого подхода? Очередная команда после ее выборки попадает в блок декодирования. Таким образом блок выборки освобождается и может выбрать следующую команду. В результате на конвейере могут находиться в различной стадии выполнения пять команд. Скорость вычисления в результате существенно возрастает.

В процессорах Pentium конвейерная архитектура была усовершенствована и получила название *суперскалярной*. В отличие от *скалярной* архитектуры i80486 (с одним конвейером), первые модели процессоров Pentium имели два конвейера. В идеале такой суперскалярный процессор должен выполнять две команды за машинный такт. Но не все так просто. Реально два конвейера Pentium не были функционально равнозначными. В связи с этим они даже имели разные названия — u-конвейер (главный) и v-конвейер (второстепенный). Главный конвейер был полнофункциональным и мог выполнять любые машинные команды. Функциональность второстепенного конвейера была ограничена основными целочисленными командами и одной командой с плавающей точкой (FXCH). Внутренняя структура обоих конвейеров такая же, как у i80486 с одним общим блоком выборки команд. Для того чтобы два разных по функциональным возможностям конвейера могли обеспечить предельную эффективность (две выполненных команды за такт работы процессора), необходимо было группировать команды из входного потока в совместимые пары. Важно заметить, что исходная последовательность команд входного потока была неизменной. Если процессору не удавалось собрать совместимую пару, то выполнялась одна команда на u-конвейере. Оставшуюся команду процессор пытался «спарить» со следующей командой входного потока.

Вернемся к процессорам микроархитектуры P6. Они имеют другую структуру конвейера. Собственно конвейера в понимании i80486 и первых Pentium уже нет. Конвейеризация заключается в том, что весь процесс обработки команд разбит на 12 стадий, которые исполняются различными

блоками процессора. Сколько именно команд обрабатывается процессором, сказать трудно. Термин *трехходовой* означает лишь то, что для исполнения из входного потока выбираются до трех команд. Известен верхний предел — в процессоре в каждый момент времени могут находиться до 30 команд в различной стадии исполнения. Детали этого процесса скрыты за понятием *динамическое исполнение с нарушением исходного порядка следования машинных команд* (out of order), что означает исполнение команд в порядке, определяемом исполнительным устройством процессора, а не исходной последовательностью команд. В основу технологии динамического исполнения положены три концепции:

Предсказание правильного адреса перехода. Основная задача механизма предсказания — исключить перезагрузку конвейера. Под переходом понимается запланированное алгоритмом изменение последовательного характера выполнения программы. Как показывает статистика, типичная программа на каждые 6–8 команд содержит 1 команду перехода. Последствия обработки перехода предсказать несложно: при наличии конвейера через каждые 6–8 команд его нужно очищать и заполнять заново в соответствии с адресом перехода. Все преимущества конвейеризации теряются. Поэтому в архитектуру Pentium в состав устройства выборки/декодирования был введен блок предсказания переходов. Вероятность правильного предсказания составляет около 80 %.

Динамический анализ потока данных. Анализ проводится с целью определения зависимостей команд программы от данных и регистров процессора с последующей оптимизацией выполнения потока команд. Главный критерий здесь — максимально полная загрузка процессора. Для реализации данного критерия допускается нарушение исходного порядка следования команд. Сбоя при этом не происходит, так как внешне логика работы программы не нарушается. Подобная внутренняя неупорядоченность исполнения команд позволяет держать процессор загруженным даже тогда, когда данные в кэш-памяти второго уровня отсутствуют и необходимо тратить время на обращение за ними в оперативную память.

Спекулятивное исполнение — способность процессора исполнять машинные команды на участках программы с условными переходами и циклами до того, как эти переходы будут разрешены алгоритмом программы. Если переход предсказан правильно, то процессор к этому моменту уже имеет исполненный код, в противном случае весь конвейер нужно очищать, загружать и исполнять код новой ветви программы, что очень накладно.

Рассмотрим порядок функционирования исполнительного устройства микроархитектуры P6 и реализацию с его помощью описанных ранее технологий. Это рассмотрение не является строгим, кое-где для лучшего понимания оно упрощено. Для иллюстрации будем использовать схему, представленную на рисунке 1.5. Из схемы видно, что структурно процессор

микроархитектуры P6 состоит из нескольких подсистем.

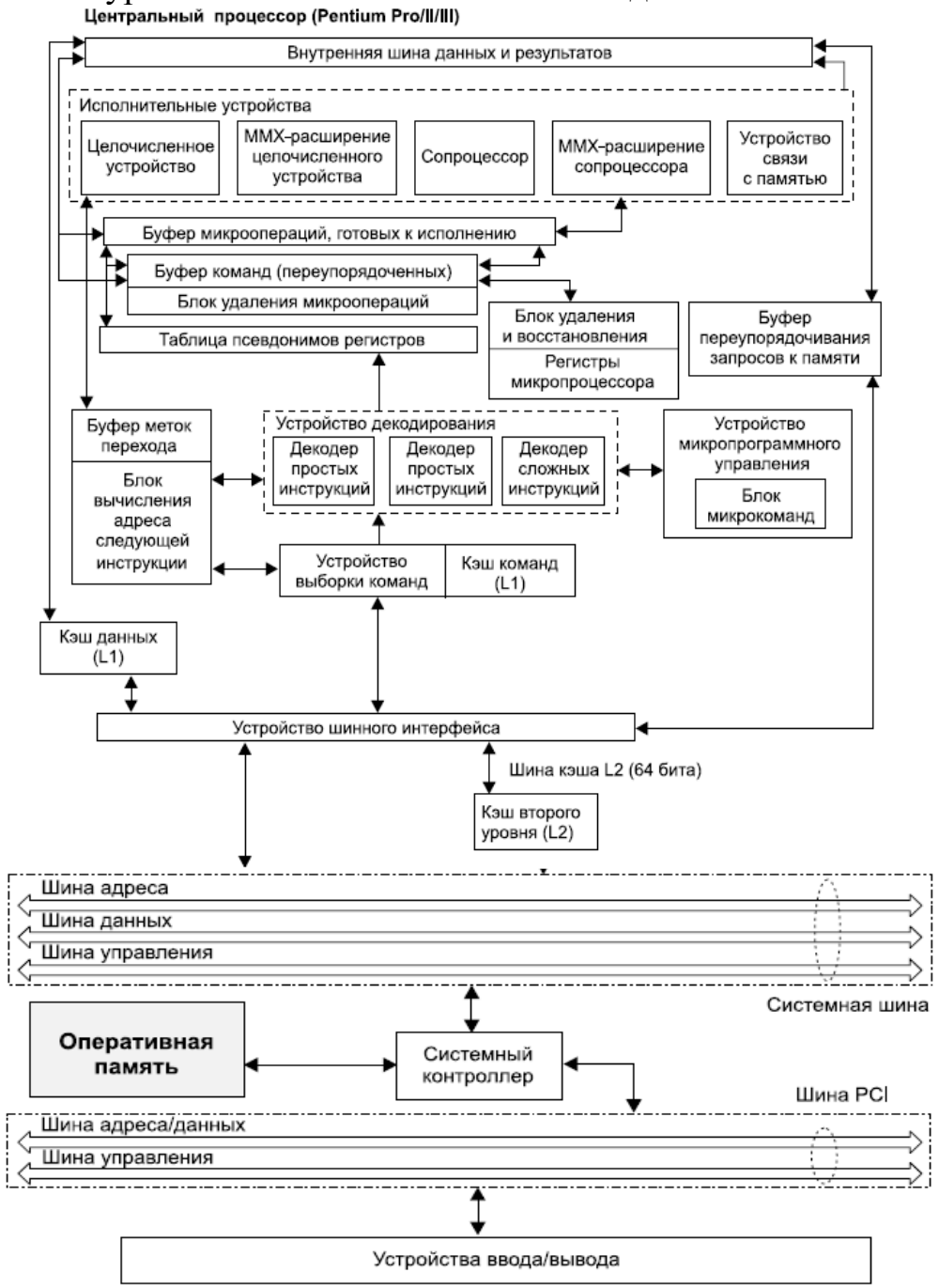


Рисунок 1.5 – Структурная схема процессора семейства P6 (Pentium Pro/II/III)

1) Подсистема памяти состоит из системной шины, кэша второго уровня L2, устройства шинного интерфейса, кэша первого уровня L1 (инструкций и данных), устройства связи с памятью и буфера переупорядочивания запросов к памяти.

2) Устройство выборки/декодирования состоит из устройства выборки команд, блока предсказания переходов, в который входят блоки меток перехода и вычисления адреса следующей инструкции, устройства декодирования, устройства микропрограммного управления и таблицы

псевдонимов регистров.

3) Буфер команд.

4) Устройство диспетчеризации/исполнения содержит буфер микроопераций, готовых к исполнению, и пять исполнительных устройств (два — для исполнения целочисленных операций, два — для исполнения операций с плавающей точкой, а также устройство связи с памятью). Необходимо заметить, что здесь допущена вольность в трактовке назначения исполнительных устройств: выделены устройства для выполнения обычных команд (целочисленных и с плавающей точкой) и MMX-команд (также целочисленных и с плавающей точкой). Реальное деление несколько иное. Такое допущение сделано исключительно с учебной целью — для более осознанного перехода от архитектуры к системе команд ассемблера.

5) Блок удаления и восстановления.

Рассмотрим подсистему памяти. Для бесперебойной работы процессора в микроархитектуре P6 используется два уровня кэш-памяти (кэширование — способ увеличения быстродействия системы за счет хранения часто используемых данных и кодов в так называемой кэш-памяти, находящейся внутри процессора (кэш-память первого уровня) либо в непосредственной близости от него (кэш-память второго уровня)). Кэш-память первого уровня состоит из кэшей команд и данных размером по 8 Кбайт, расположенных внутри процессора в непосредственной близости к его исполнительной части. Кэш-память второго уровня является внешней по отношению к процессору (но в едином конструктиве с ним), имеет значительно больший размер (256 Кбайт, 512 Кбайт или 1 Мбайт) и соединена с ядром процессора посредством 64-разрядной шины. Разделение кэш-памяти на две части (для кода и данных) обеспечивает бесперебойную поставку машинных инструкций и элементов данных в исполнительное устройство процессора. Исходные данные для кэш-памяти первого уровня предоставляет кэш-память второго уровня. Заметьте, что информация из нее поступает на устройство шинного интерфейса и далее в соответствующую кэш-память первого уровня по 64-разрядной шине. При этом благодаря более быстрому обновлению содержимого кэш-памяти первого уровня обеспечивается высокий темп работы процессора.

Устройство шинного интерфейса обращается к оперативной памяти системы через внешнюю системную шину. Эта 64-разрядная шина ориентирована на обработку запросов, то есть каждый шинный запрос обрабатывается отдельно и требует обратной реакции. Пока устройство шинного интерфейса ожидает ответа на один запрос шины, возможно формирование многочисленных дополнительных запросов. Все они обслуживаются в порядке поступления. Считываемые по запросу данные помещаются в кэш второго уровня. То есть процессор посредством устройства шинного интерфейса читает команды и данные из кэша второго уровня. Устройство шинного интерфейса взаимодействует с кэшем второго уровня

через 64разрядную шину кэша, которая также ориентирована на обработку запросов и функционирует на тактовой частоте процессора. Доступ к кэшу первого уровня осуществляется через внутренние шины на тактовой частоте процессора. Синхронная работа с системной памятью кэш-памяти обоих уровней осуществляется благодаря специальному протоколу MESI.

Запросы от команд на получение операндов из памяти в исполнительном устройстве процессора обслуживаются посредством *устройства связи с памятью* и *буфера переупорядочивания запросов к памяти*. Эти два устройства специально включены в схему для того, чтобы обеспечить бесперебойное снабжение исполняемых команд необходимыми данными. Особо стоит подчеркнуть роль буфера переупорядочивания запросов к памяти. Он отслеживает все запросы к операндам в памяти и выполняет функции планирующего устройства. Если нужные для очередной операции данные в кэш-памяти первого уровня (L1) отсутствуют, то буфер переупорядочивания запросов к памяти автоматически передает информацию о неудачном обращении к данным кэшу второго уровня (L2). Если и в кэше L2 нужных данных не оказалось, то буфер переупорядочивания запросов к памяти заставляет устройство шинного интерфейса сформировать запрос к оперативной памяти компьютера.

Устройство выборки/декодирования извлекает одну 32байтную строку кэша команд (L1) за такт и передает в декодер, который преобразует ее в последовательность микроопераций. Поток микроопераций (пока он еще соответствует последовательности исходных команд) поступает в *буфер команд*. *Устройство выборки команд* вычисляет указатель на следующую команду, подлежащую выборке, на основании информации трех источников: буфера меток перехода, состояния прерывания/исключения и сообщения от исполнительного целочисленного устройства об ошибке в предсказании метки перехода. Важная часть этого процесса — предсказание метки перехода, которое выполняется по специальному алгоритму. В его основе лежит работа с *буфером меток перехода*, который содержит информацию о последних 256 переходах. Если очередная команда, выбираемая из памяти, является командой перехода, то содержащийся в ней адрес перехода сравнивается с адресами, уже находящимися в буфере меток перехода. Если этот адрес уже есть в буфере меток переходов, то он станет адресом следующей команды, с которой устройство выборки будет извлекать очередную команду. Если искомого адреса перехода в буфере нет, то выборка команд из памяти будет продолжена до момента исполнения команды перехода исполнительным устройством. В результате ее исполнения становится ясно, было ли правильным решение об адресе начала выборки следующих команд после выборки команды перехода. Если предсказанный переход оказывается неверным, то конвейер сбрасывается и загружается заново в соответствии с адресом перехода. Цель предсказания переходов — в том, чтобы устройство исполнения постоянно было занято полезной работой и сброс конвейера производился как можно реже.

Устройство выборки команд выбирает команды для исполнения и помещает их в устройство декодирования. Устройство декодирования состоит из трех параллельно работающих *декодеров* (два простых и один сложный). Именно эти декодеры воплощают в жизнь понятие исполнения с нарушением исходного порядка следования команд (out of order) и являются теми самыми тремя входами (three way) в исполнительное устройство процессора. Декодеры преобразуют команды процессора в микрооперации. Микрооперации представляют собой примитивные команды, которые выполняются пятью *исполнительными устройствами* процессора, работающими параллельно. Многие машинные команды преобразуются в одиночные микрооперации (это делает простой декодер), а некоторые машинные команды — в последовательность от двух и более (оптимально — четырех) микроопераций (это делает сложный декодер). Информация о последовательности микроопераций для реализации конкретной машинной команды содержится в *устройстве микропрограммного управления*. Кроме команд, декодеры обрабатывают также префиксы команд. Декодер команд может формировать до шести микроопераций за такт — по одной от простых декодеров и до четырех от сложного декодера. Для достижения наибольшей производительности работы декодеров необходимо, чтобы на их вход поступали команды, которые декодируются шестью микрооперациями в последовательности 4 + 1 + 1. Если время работы программы критично, то имеет смысл провести ее оптимизацию, заключающуюся в переупорядочивании исходного набора команд таким образом, чтобы группы команд формировали последовательности микроопераций по схеме 4 + 1 + 1. После того как команды разбиты на микрооперации, порядок их выполнения трудно предсказать. При этом могут возникнуть проблемы с таким критичным ресурсом, как регистры. Суть здесь в том, что если в двух соседних фрагментах программы данные помещались в одинаковые регистры, откуда они, возможно, записывались в некоторые области памяти, а после переупорядочивания эти фрагменты перемешались, то как разобраться в том, какие регистры и где использовались. Эта проблема носит название *проблемы ложных взаимозависимостей* и решается с помощью *механизма переименования регистров*. Основу этого механизма составляет набор из 40 внутренних универсальных регистров, которые и задействуются в реальных вычислениях исполнительным устройством абсолютно прозрачно для программ. Универсальные регистры могут работать как с целыми числами, так и со значениями с плавающей точкой. Информация о действительных именах регистров процессора и их внутренних именах (номерах универсальных регистров) помещается в *таблицу псевдонимов регистров*.

В заключение процесса декодирования *устройство управления таблицей псевдонимов регистров* добавляет к микрооперациям биты состояния и флаги, чтобы подготовить их к неупорядоченному выполнению, после чего посылает получившиеся микрооперации в *буфер переупорядоченных команд*. Нужно заметить, что новый порядок их следования не соответствует порядку

следования соответствующих команд в исходной программе. Буфер переупорядоченных команд представляет собой массив ассоциативной памяти, физически выполненный в виде 40 регистров и представляющий собой кольцевую структуру, элементы которой содержат два типа микроопераций: ожидающие своей очереди на исполнение и уже частично выполненные, но не до конца из-за их переупорядочивания и зависимости от других частично или полностью не выполненных микроопераций.

Устройство диспетчеризации/исполнения может выбирать микрооперации из этого буфера в любом порядке.

Устройство диспетчеризации/исполнения планирует и исполняет неупорядоченную последовательность микроопераций из буфера переупорядоченных команд. Но оно не занимается непосредственной выборкой микроопераций из буфера переупорядоченных команд, так как в нем могут содержаться и не готовые к исполнению микрооперации. Этим занимается устройство, управляющее специальным буфером, который условно назовем *буфером микроопераций, готовых к исполнению*. Оно постоянно сканирует буфер переупорядоченных команд в поисках микроопераций, готовых к исполнению (фактически это означает доступность всех операндов), после чего посылает их соответствующим исполнительным устройствам, если они не заняты. Результаты исполнения микроопераций возвращаются в буфер переупорядоченных команд и сохраняются там наряду с другими микрооперациями до тех пор, пока не будут удалены *устройством удаления и восстановления*.

Подобная схема планирования и исполнения программ реализует классический принцип неупорядоченного выполнения, при котором микрооперации посылаются исполнительным устройствам вне зависимости от их расположения в исходном алгоритме. В случае, если к выполнению одновременно готовы две или более микрооперации одного типа (например, целочисленные), то они выполняются в соответствии с принципом FIFO (First In, First Out — первым пришел, первым ушел), то есть в порядке поступления в буфер переупорядоченных команд.

Непосредственно исполнительное устройство состоит из пяти блоков, каждый из которых обрабатывает свой тип микроопераций: два целочисленных устройства, два устройства для вычислений с плавающей точкой и одно устройство связи с памятью. Таким образом, за один машинный такт одновременно исполняется пять микроопераций.

Два целочисленных исполнительных устройства могут параллельно обрабатывать две целочисленные микрооперации. Одно из этих целочисленных исполнительных устройств специально предназначено для работы с микрооперациями переходов. Оно способно обнаружить непредсказанный переход и сообщить об этом устройству выборки команд, чтобы перезапустить конвейер. Такая операция реализована следующим образом. Декодер команд отмечает каждую микрооперацию перехода и адрес перехода. Когда целочисленное исполнительное устройство выполняет

микрооперацию перехода, то оно определяет, был предсказан переход или нет. Если переход предсказан правильно, то микрооперация отмечается пригодной для использования, и выполнение продолжается по предсказанной ветви. Если переход предсказан неправильно, то целочисленное исполнительное устройство изменяет состояние всех последующих микроопераций с тем, чтобы удалить их из буфера переупорядоченных команд. После этого целочисленное устройство помещает метку перехода в буфер меток перехода, который, в свою очередь, совместно с устройством выборки команд перезапускает конвейер относительно нового исполнительного адреса.

Устройство связи с памятью управляет загрузкой и сохранением данных для микроопераций. Для их загрузки в исполнительное устройство достаточно определить только адрес памяти, поэтому такое действие кодируется одной микрооперацией. Для сохранения данных необходимо определять и адрес, и записываемые данные, поэтому это действие кодируется двумя микрооперациями. Та часть устройства связи с памятью, которая управляет сохранением данных, имеет два блока, позволяющие ему обрабатывать адрес и данные для микрооперации параллельно. Это позволяет устройству связи с памятью выполнить загрузку и сохранение данных для микроопераций параллельно в одном такте.

Исполнительные устройства с плавающей точкой аналогичны устройствам в более ранних моделях процессора Pentium. Было добавлено только несколько новых команд с плавающей точкой для организации условных переходов и перемещений.

Последний блок в этой схеме выполнения команд исходной программы — *блок удаления и восстановления*, задачей которого является возврат вычислительного процесса в рамки, определенные исходной последовательностью команд. Для этого он постоянно сканирует буфер переупорядоченных команд на предмет обнаружения полностью выполненных микроопераций, не имеющих связи с другими микрооперациями. Такие микрооперации удаляются из буфера переупорядоченных команд и восстанавливаются в порядке, соответствующем порядку следования команд исходной программы с учетом прерываний, исключений, точек прерывания и переходов. Блок удаления и восстановления может удалить три микрооперации за один машинный такт. При восстановлении порядка следования команд блок удаления и восстановления записывает результаты в реальные регистры процессора и в оперативную память.

Таким образом, система динамического исполнения команд позволяет организовать прохождение команд программы через исполнительное устройство процессора эффективнее, чем это было в конвейере процессора i80486 и первых процессоров Pentium.

5.3.2 Микроархитектура NetBurst

Микроархитектура NetBurst, реализованная в процессоре Pentium IV, является развитием идей микроархитектуры P6, поэтому рассматривается довольно конспективно. Судя по названию (net — сеть, burst — прорыв), микроархитектура NetBurst призвана обеспечить некий «сетевой прорыв». Очевидно, что этим разработчики хотели подчеркнуть те новые особенности процессора Pentium IV, которые позволяют организовать более быструю и эффективную работу приложений в современных сетевых и мультимедийных информационных средах. Отметим наиболее важные свойства новой микроархитектуры.

Быстрая исполнительная часть процессора. АЛУ процессора работает на удвоенной частоте процессора. За каждый такт процессора выполняются две основные целочисленные команды. Обеспечена более высокая пропускная способность потока команд через исполнительную часть процессора и уменьшены различные задержки.

Гиперконвейерная технология. Гиперконвейер Pentium IV состоит из 20 ступеней. Цель увеличения длины конвейера — упрощение задач, реализуемых каждой из его ступеней, и, как следствие, упрощение соответствующей аппаратной логики.

Улучшенная технология динамического исполнения благодаря более глубокой «произвольности» в порядке исполнения кода и усовершенствованной системе предсказания переходов. Размер буфера меток перехода увеличен до 4 Кбайт (Pentium III — 512 байт). Усовершенствован и сам алгоритм предсказания. В результате вероятность предсказания перехода возрастает до 95 %.

Новая подсистема кэширования. Отсутствует кэш команд первого уровня. Вместо него введен кэш трасс. Трассами называются последовательности микроопераций, в которые были декодированы ранее выбранные команды. Кэш трасс способен хранить до 12 Кбайт микроопераций и доставлять исполнительному ядру до 3 микроопераций за такт. Кэш второго уровня работает на полной частоте ядра процессора.

Структурная схема процессора Pentium IV показана на рисунке 1.6. Микроархитектура NetBurst поддерживает еще одну новую технологию — Hyper Threading. Данная технология позволяет на базе одного физического процессора Pentium IV моделировать несколько логических, каждый из которых имеет собственное архитектурное пространство IA-32. Под архитектурным пространствомIA-32 понимается совокупность регистров данных, сегментных регистров, системных регистров и регистров MSR. Каждый логический процессор имеет также собственный контроллер прерываний APIC.

На этом, наверное, следует завершить обсуждение общих вопросов, связанных с архитектурой процессоров Intel. Задача данного комплекса — разобраться, как управлять этими сложными процессорами. Программисту совершенно необязательно помнить о схемотехнических тонкостях

реализации процессора, ему достаточно знать, какие части процессора ему доступны и как с ними взаимодействовать для реализации некоторой задачи.

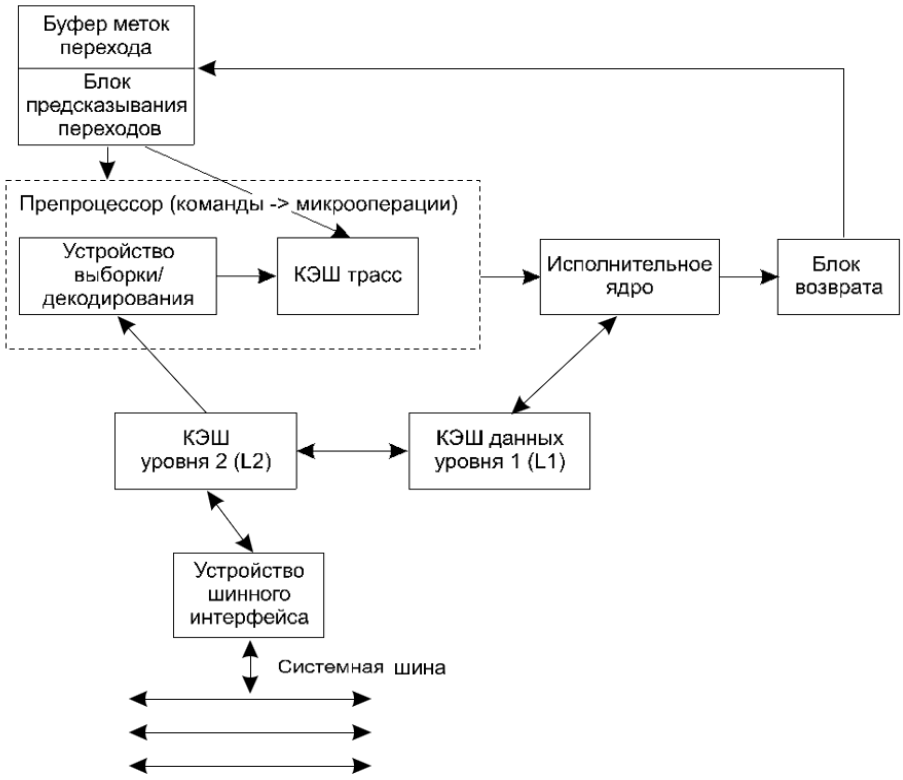


Рис. 1.6 – Структурная схема процессора Pentium IV

Первый шаг в этом направлении — знакомство с *программной моделью процессора*. Эта модель описывает видимые для программиста объекты архитектуры процессора, знание которых позволяет программисту эффективно и в полном объеме использовать возможности процессора.

5.3.3 Программная модель IA-32

Любая выполняющаяся программа получает в свое распоряжение определенный набор ресурсов процессора. Эти ресурсы необходимы для обработки и хранения в памяти команд и данных программы, а также информации о текущем состоянии программы и процессора. Программную модель процессора в архитектуре IA-32 процессоров Intel составляет следующий набор ресурсов (рисунок 1.7):

- пространство адресуемой памяти до 232 – 1 байт (4 Гбайт), для Pentium III/IV — до 236 – 1 байт (64 Гбайт);
- набор регистров для хранения данных общего назначения; набор сегментных регистров;
- набор регистров состояния и управления;
- набор регистров устройства вычислений с плавающей точкой (сопроцессора);
- набор регистров целочисленного MMX-расширения, отображенных на регистры сопроцессора (впервые появились в архитектуре процессора

Pentium MMX);

- набор регистров MMX-расширения с плавающей точкой (впервые появились в архитектуре процессора Pentium III);

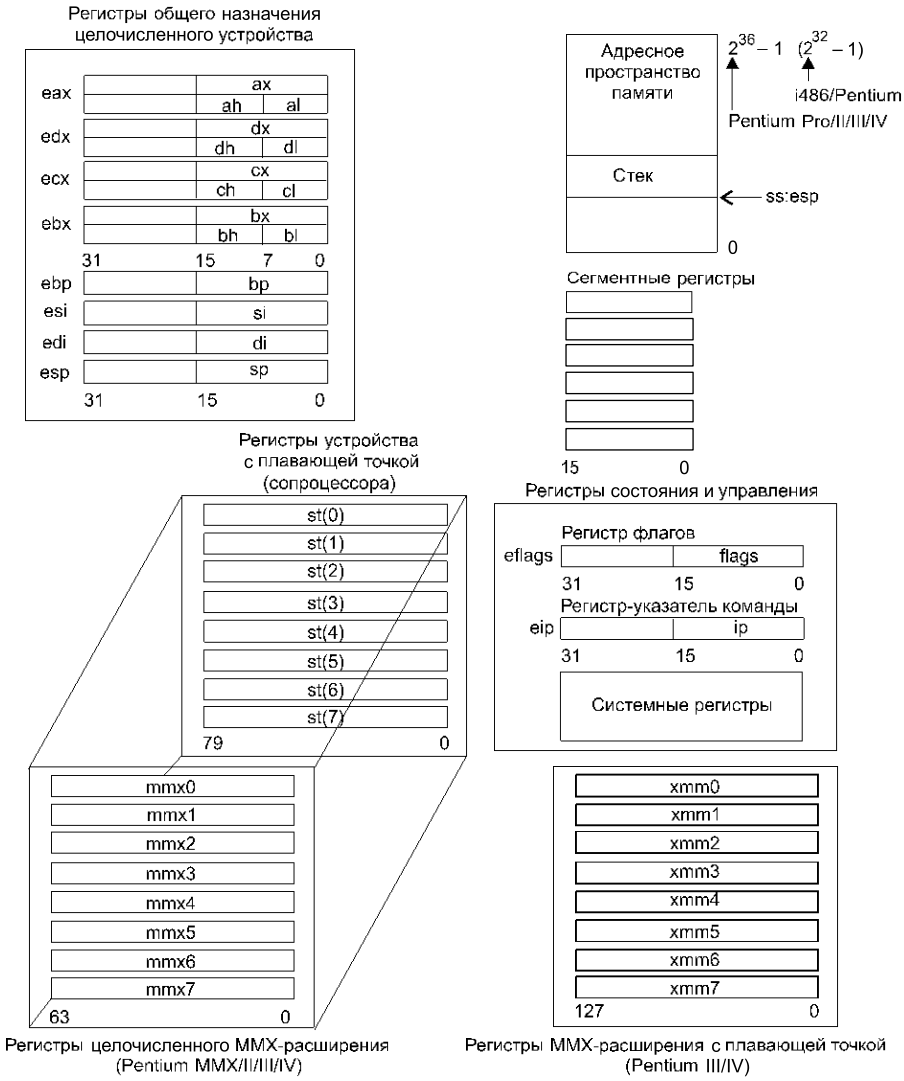


Рисунок 1.7 – Программная модель архитектуры IA-32 процессоров Intel

- программный стек — специальная информационная структура, работа с которой предусмотрена на уровне машинных команд (более подробно она будет обсуждена позже).

Это основной набор ресурсов. Кроме того, к ресурсам, поддерживаемым архитектурой IA-32, необходимо отнести порты ввода-вывода, счетчики мониторинга производительности.

Программные модели более ранних процессоров (i486, первые Pentium) отличаются меньшим размером адресуемого пространства оперативной памяти (2^{32} – 1, так как разрядность их шины адреса составляет 32 бита) и

отсутствием некоторых групп регистров. Для каждой группы регистров в скобках показано, начиная с какой модели данная группа регистров появилась в программной модели процессоров Intel. Если такого обозначения нет, то это означает, что данная группа регистров присутствовала в процессорах i386 и i486. Что касается еще более ранних процессоров i8086/88, то на самом деле они тоже полностью представлены на схеме, но составляют лишь ее небольшую ее часть. В программную модель данных процессоров входят 8- и 16-разрядные регистры общего назначения, сегментные регистры, регистры FLAGS, IP и адресное пространство памяти размером до 1Мбайт. Свойства некоторых перечисленных далее программно-доступных ресурсов определяются текущим режимом работы процессора.

Режимы работы процессора архитектуры IA-32. Режим работы процессора определяет поведение, номенклатуру и свойства доступных ресурсов процессора. Перевод процессора из одного режима в другой осуществляется специальными программными и аппаратными методами.

В рамках архитектуры IA-32 доступны следующие режимы работы процессора.

Режим реальных адресов, или просто *реальный режим* (real mode) — это режим, в котором работал i8086. Наличие его в i486 и Pentium обусловлено тем, что фирма Intel старается обеспечить в новых моделях процессоров возможность функционирования программ, разработанных для ранних моделей.

Защищенный режим (protected mode) позволяет максимально реализовать все идеи, заложенные в процессорах архитектуры IA-32, начиная с i80286. Программы, разработанные для i8086 (реального режима), не могут функционировать в защищенном режиме. Одна из причин этого связана с особенностями формирования физического адреса в защищенном режиме.

Режим виртуального процессора 8086 предназначен для организации многозадачной работы программ, разработанных для реального режима (процессора i8086), совместно с программами защищенного режима. Переход в этот режим (virtual 8086 mode) возможен, если процессор уже находится в защищенном режиме. Работа программ реального режима в режиме виртуального i8086 возможна благодаря тому, что процесс формирования физического адреса для них производится по правилам реального режима.

Режим системного управления (System Management Mode, SMM) — это новый режим работы процессора, впервые появившийся в процессоре Pentium. Он обеспечивает операционную систему механизмом для выполнения машинно-зависимых функций, таких как перевод компьютера в режим пониженного энергопотребления или выполнения действий по защите системы. Для перехода в данный режим процессор должен получить специальный сигнал SMI от усовершенствованного программируемого контроллера прерываний (Advanced Programmable Interrupt Controller,

APIC), при этом сохраняется состояние вычислительной среды процессора. Функционирование процессора в этом режиме подобно его работе в режиме реальных адресов. Возврат из этого режима производится специальной командой процессора.

Процессор всегда начинает работу в реальном режиме.

Набор регистров. *Регистрами* называются области высокоскоростной памяти, расположенные внутри процессора в непосредственной близости от его исполнительного ядра. Доступ к ним осуществляется несравнимо быстрее, чем к ячейкам оперативной памяти. Соответственно, машинные команды с операндами в регистрах выполняются максимально быстро, поэтому в программах на языке ассемблера регистры используются очень интенсивно. К сожалению, архитектура IA-32 предоставляет в распоряжение программиста не слишком много регистров, поэтому они являются критически важным ресурсом и за их содержимым приходится следить очень внимательно.

Большинство регистров имеют определенное функциональное назначение. С точки зрения программиста, их можно разделить на две большие группы.

Первую группу образуют пользовательские регистры, к которым относятся:

- регистры общего назначения EAX/AX/AH/AL, EBX/BX/BH/BL, EDX/DX/DH/DL, ECX/ CX/CH/CL, EBP/BP, ESI/SI, EDI/DI, ESP/SP предназначены для хранения данных и адресов, программист может их использовать (с определенными ограничениями) для реализации своих алгоритмов;
- сегментные регистры CS, DS, SS, ES, FS, GS используются для хранения адресов сегментов в памяти;
- регистры сопроцессора ST(0), ST(1), ST(2), ST(3), ST(4), ST(5), ST(6), ST(7) предназначены для написания программ, использующих тип данных с плавающей точкой;
- целочисленные регистры MMX-расширения MMX0, MMX1, MMX2, MMX3, MMX4, MMX5, MMX6, MMX7;
- регистры MMX-расширения с плавающей точкой XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6, XMM7;
- регистры состояния и управления (регистр флагов EFLAGS/FLAGS и регистр-указатель команды EIP/IP) содержат информацию о состоянии процессора, исполняемой программы и позволяют изменить это состояние.

Во вторую группу входят системные регистры, то есть регистры, предназначенные для поддержания различных режимов работы, сервисных функций, а также регистры, специфичные для определенной модели процессора. Перечислим системные регистры, поддерживаемые IA-32:

- управляющие регистры CR0…CR4 определяют режим работы процессора и характеристики текущей исполняемой задачи;

- регистры управления памятью GDTR, IDTR, LDTR и TR используются в защищенном режиме работы процессора для локализации управляющих структур этого режима;

- отладочные регистры DR0…DR7 предназначены для мониторинга и управления различными аспектами отладки;

- регистры типов областей памяти MTRR используются для аппаратного управления кэшированием в целях назначения соответствующих свойств областям памяти;

- машинно-зависимые регистры MSR используются для управления процессором, контроля за его производительностью, получения информации об ошибках.

Почему в обозначениях многих из регистров общего назначения присутствует наклонная разделительная черта? Это не разные регистры — это части одного большого 32-разрядного регистра, но их можно использовать в программе как отдельные объекты. Зачем так сделано? Чтобы обеспечить работоспособность программ, написанных для прежних 16-разрядных моделей процессоров фирмы Intel начиная с i8086. Процессоры i486 и Pentium имеют в основном 32-разрядные регистры.

Их количество, за исключением сегментных регистров, такое же, как и у i8086, но размерность больше, что и отражено в обозначениях — они имеют приставку e (extended).

Регистры общего назначения. Регистры общего назначения используются в программах для хранения:

- операндов логических и арифметических операций;
- компонентов адреса;
- указателей на ячейки памяти.

В принципе, все эти регистры доступны для хранения операндов без особых ограничений, хотя в определенных условиях некоторые из них имеют жесткое функциональное назначение, закрепленное на уровне логики работы машинных команд. Среди всех этих регистров особо следует выделить регистр ESP. Его не следует использовать явно для хранения каких-либо операндов программы, так как в нем хранится указатель на вершину стека программы.

Все регистры этой группы позволяют обращаться к своим «младшим» частям (см. рисунок 1.7). Обращение производится по их именам. Важно отметить, что использовать для самостоятельной адресации можно только младшие 16- и 8-разрядные части этих регистров. Старшие 16 битов как самостоятельные объекты недоступны. Это сделано, как мы отметили ранее, для совместимости с первыми 16-разрядными моделями процессоров фирмы Intel. Перечислим регистры, относящиеся к группе регистров общего назначения и физически находящиеся в процессоре внутри арифметико-логического устройства (поэтому их еще называют *регистрами АЛУ*):

регистр-аккумулятор (Accumulator register) EAX/AX/AH/AL

применяется для хранения промежуточных данных, в некоторых командах его использование обязательно;

базовый регистр (Base register) EBX/BX/BH/BL применяется для хранения базового адреса некоторого объекта в памяти;

регистр-счетчик (Count register) ECX/CX/CH/CL применяется в командах, производящих некоторые повторяющиеся действия. Использование регистра-счетчика зачастую скрыто в алгоритме работы той или иной команды. Например, команда организации цикла LOOP помимо передачи управления анализирует и уменьшает на единицу значение регистра ECX/CX;

регистр данных (Data register) EDX/DX/DH/DL, так же как и регистр EAX/AX/AH/ AL, хранит промежуточные данные (в некоторых командах его явное использование обязательно, в других он используется неявно).

Следующие два регистра предназначены для поддержки так называемых цепочечных операций, то есть операций, производящих последовательную обработку цепочек элементов, каждый из которых может иметь длину 32, 16 или 8 бит:

регистр индекса источника (Source Index register) ESI/SI в цепочечных операциях содержит текущий адрес элемента в цепочке-источнике;

регистр индекса приемника (Destination Index register) EDI/DI в цепочечных операциях содержит текущий адрес в цепочке-приемнике.

В архитектуре процессора на программно-аппаратном уровне поддерживается такая структура данных, как *стек*. Для работы со стеком в системе команд процессора есть специальные команды, а в программной модели процессора для этого существуют специальные регистры:

регистр указателя стека (Stack Pointer register) ESP/SP содержит указатель на вершину стека в текущем сегменте стека;

регистр указателя базы кадра стека (Base Pointer register) EBP/BP предназначен для организации произвольного доступа к данным внутри стека.

Не стоит пугаться столь жесткого функционального назначения регистров АЛУ. На самом деле при программировании хранить операнды команд можно в большинстве регистров, причем практически в любых сочетаниях. Однако всегда следует помнить, что некоторые команды требуют строго определенных регистров. Цель подобного жесткого закрепления регистров для этих команд — более компактная кодировка их машинного представления. Знание особенностей использования регистров машинными командами (см. приложение) позволяет, при необходимости, экономить память, занимаемую кодом программы, и более эффективно программировать алгоритм.

Сегментные регистры. Процессоры Intel аппаратно поддерживают *сегментную* организацию программы. Это означает, что любая программа состоит из трех сегментов: кода, данных и стека. Логически машинные

команды в архитектуре IA-32 построены так, что при выборке каждой команды для доступа к данным программы или к стеку неявно используется информация из вполне определенных сегментных регистров. В зависимости от режима работы процессора по их содержимому определяются адреса памяти, с которых начинаются соответствующие сегменты. В программной модели IA-32 имеется шесть *сегментных регистров* CS, SS, DS, ES, GS, FS, служащих для доступа к четырем типам сегментов.

Сегмент кода содержит команды программы. Для доступа к этому сегменту служит *регистр сегмента кода* (Code Segment register) CS. Он содержит адрес сегмента с машинными командами, к которому имеет доступ процессор (эти команды загружаются в конвейер процессора).

Сегмент данных содержит обрабатываемые программой данные. Для доступа к этому сегменту служит *регистр сегмента данных* (Data Segment register) DS, который хранит адрес сегмента данных текущей программы.

Сегмент стека представляет собой область памяти, называемую стеком. Работу со стеком процессор организует по следующему принципу: последний записанный в эту область элемент выбирается первым. Для доступа к этой области служит *регистр сегмента стека* (Stack Segment register) SS, содержащий адрес сегмента стека.

Дополнительный сегмент данных. Неявно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в регистре сегмента данных DS. Если программе недостаточно одного сегмента данных, то она имеет возможность задействовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в регистре DS, при использовании дополнительных сегментов данных их адреса требуется указывать явно с помощью специальных префиксов переопределения сегментов в команде. Адреса дополнительных сегментов данных должны содержаться в *регистрах дополнительного сегмента данных* (Extension Data Segment registers) ES, GS, FS.

Регистры состояния и управления. В процессор включены два регистра (см. рисунок 1.6), постоянно содержащие информацию о состоянии как самого процессора, так и программы, команды которой он в данный момент обрабатывает:

- регистр-указатель команд EIP/IP;
- регистр флагов EFLAGS/FLAGS.

С помощью этих регистров можно также ограниченным образом управлять состоянием процессора.

Регистр-указатель команд (Instruction Pointer register) EIP/IP имеет разрядность 32(16) бита и содержит смещение следующей подлежащей выполнению команды относительно содержимого регистра сегмента кода CS в текущем сегменте команд. Этот регистр непосредственно недоступен программисту, но загрузка и изменение его значения производятся

различными командами управления, к которым относятся команды условных и безусловных переходов, вызова процедур и возврата из процедур. Возникновение прерываний также приводит к модификации регистра EIP/IP.

Разрядность *регистра флагов* (flag register) EFLAGS/FLAGS равна 32(16) битам. Отдельные биты данного регистра имеют определенное функциональное назначение и называются *флагами*. Младшая часть регистра EFLAGS/FLAGS полностью аналогична регистру FLAGS процессора i8086. На рисунке 1.8 показано содержимое регистра EFLAGS.

Исходя из особенностей использования, флаги регистра EFLAGS/FLAGS можно разделить на три группы.

В первую группу флагов регистра EFLAGS/FLAGS входят 8 *флагов состояния*. Эти флаги могут изменяться после выполнения машинных команд. Флаги состояния регистра EFLAGS отражают особенности результата исполнения арифметических или логических операций. Это дает возможность анализировать состояние вычислительного процесса и реагировать на него с помощью команд условных переходов и вызовов подпрограмм.

Флаг переноса (carry flag) CF: 1 — арифметическая операция произвела перенос из старшего бита результата, старшим является 7-й, 15-й или 31-й бит в зависимости от размерности операнда; 0 — переноса не было.

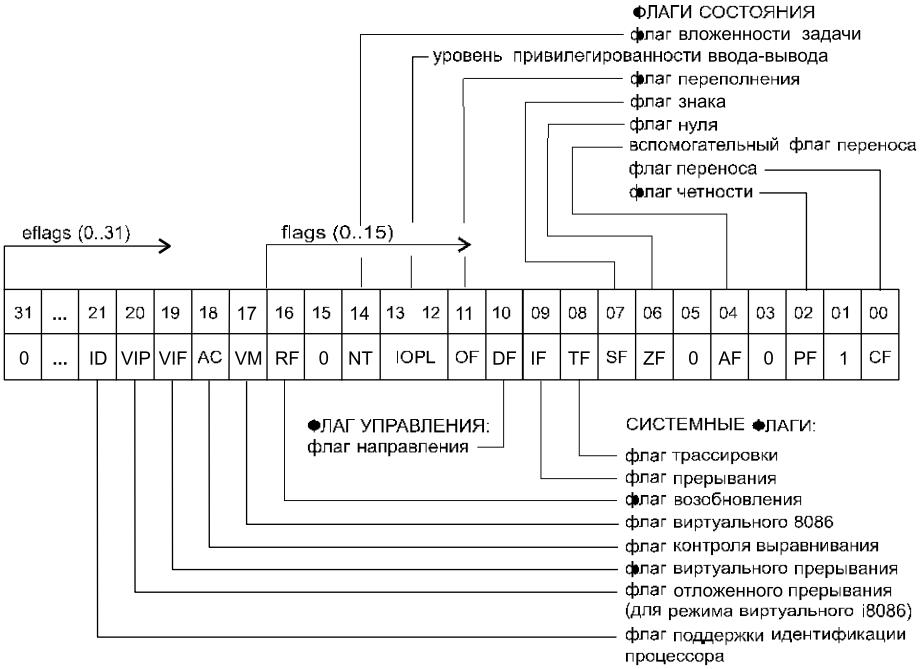


Рисунок 1.8 – Содержимое регистра eflags

Флаг четности (parity flag) PF: 1 — 8 младших разрядов (этот флаг только для 8 младших разрядов операнда любого размера) результата содержат четное число единиц; 0 — 8 младших разрядов результата содержат

нечетное число единиц.

Вспомогательный флаг переноса (auxiliary carry flag) AF применяется только для команд, работающих с BCD-числами. Фиксирует факт заема из младшей тетрады результата: 1 — в результате операции сложения был произведен перенос из разряда 3 в старший разряд или при вычитании был заем в разряд 3 младшей тетрады из значения в старшей тетраде; 0 — переносов и заемов в третий разряд (из третьего разряда) младшей тет-рады результата не было.

Флаг нуля (zero flag) ZF: 1 — результат нулевой; 0 — результат ненулевой.

Флаг знака (sign flag) SF отражает состояние старшего бита результата (биты 7, 15 или 31 для 8-, 16- или 32-разрядных операндов соответственно): 1 — старший бит результата равен 1; 0 — старший бит результата равен 0.

Флаг переполнения (overflow flag) OF используется для фиксации факта потери значащего бита при арифметических операциях: 1 — в результате операции происходит перенос в старший знаковый бит результата или заем из старшего знакового бита результата (биты 7, 15 или 31 для 8-, 16- или 32-разрядных операндов соответственно); 0 — в результате операции не происходит переноса в старший знаковый бит результата или заема из старшего знакового бита результата.

Уровень привилегированности ввода-вывода (Input/Output privilege level) IOPL используется в защищенном режиме работы процессора для контроля доступа к командам ввода-вывода в зависимости от привилегированности задачи.

Флаг вложенности задачи (nested task) NT используется в защищенном режиме работы процессора для фиксации того факта, что одна задача вложена в другую.

Во вторую группу флагов (группа флагов управления) регистра EFLAGS/FLAGS входит всего один *флаг направления* (directory flag) DF. Он находится в десятом бите регистра EFLAGS и используется цепочечными командами. Значение флага DF определяет направление поэлементной обработки в этих операциях: от начала строки к концу (DF = 0) либо, наоборот, от конца строки к ее началу (DF = 1). Для работы с флагом DF существуют специальные команды CLD (снять флаг DF) и STD (установить флаг DF). Применение этих команд позволяет привести флаг DF в соответствие с алгоритмом и обеспечить автоматическое увеличение или уменьше-ние счетчиков при выполнении операций со строками.

В третью группу флагов регистра EFLAGS/FLAGS входит 8 *системных флагов*, управляющих вводом-выводом, маскируемыми прерываниями, отладкой, переключением между задачами и режимом виртуального процессора 8086. Прикладным программам не рекомендуется модифицировать без необходимости эти флаги, так как в большинстве случаев это ведет к прерыванию работы программы. Далее перечислены системные флаги и их назначение.

Флаг трассировки (trace flag) TF предназначен для организации

пошаговой работы процессора.

Флаг прерывания (interrupt enable flag) IF предназначен для разрешения или запрещения (маскирования) аппаратных прерываний (прерываний по входу INTR).

Флаг возобновления (resume flag) RF используется при обработке прерываний от регистров отладки.

Флаг режима виртуального процессора 8086 (virtual 8086 mode) VM является признаком работы процессора в режиме виртуального 8086.

Флаг контроля выравнивания (alignment check) AC предназначен для разрешения контроля выравнивания при обращениях к памяти.

Флаг виртуального прерывания (virtual interrupt flag) VIF.

Флаг отложенного виртуального прерывания (virtual interrupt pending flag) VIP.

Флаг идентификации (identification flag) ID используется для того, чтобы показать факт поддержки процессором инструкции CPUID. Если программа может установить или сбросить этот флаг, это означает, что данная модель процессора поддерживает инструкцию CPUID.

6 Организация памяти

Физическая память, к которой процессор имеет доступ по шине адреса (см. рис. 1.1), называется *оперативной памятью* (или оперативным запоминающим устройством — ОЗУ). На самом нижнем уровне память компьютера можно рассматривать как массив *битов*. Один бит может хранить значение 0 или 1. Для физической реализации битов и работы с ними идеально подходят логические схемы. Но процессору неудобно работать с памятью на уровне битов, поэтому реально ОЗУ организовано как последовательность ячеек — *байтов*. Один байт состоит из восьми битов. Каждому байту соответствует свой уникальный адрес (его номер), называемый *физическим*. Диапазон значений физических адресов зависит от разрядности шины адреса процессора. Для i486 и Pentium он находится в пределах от 0 до $2^{32} – 1$ (4 Гбайт). Для процессоров Pentium Pro/II/III/IV этот диапазон шире — от 0 до $2^{36} – 1$ (64 Гбайт).

Механизм управления памятью полностью аппаратный. Это означает, что программа не может сама сформировать физический адрес памяти на адресной шине. Ей приходится «играть» по правилам процессора. Что это за правила, мы узнаем чуть позже. Пока же отметим, что в конечном итоге этот механизм позволяет обеспечить:

- компактность хранения адреса в машинной команде;
- гибкость механизма адресации;
- защиту адресных пространств задач в многозадачной системе;
- поддержку виртуальной памяти.

Процессор аппаратно поддерживает две модели использования

оперативной памяти.

В *сегментированной модели* программе выделяются непрерывные области памяти (сегменты), а сама программа может обращаться только к данным, которые находятся в этих сегментах.

Страничную модель можно рассматривать как надстройку над сегментированной моделью. В случае использования этой модели оперативная память рассматривается как совокупность блоков фиксированного размера (4 Кбайт и более). Основное применение этой модели связано с организацией виртуальной памяти, что позволяет операционной системе использовать для работы программ пространство памяти большее, чем объем физической памяти. Для процессоров i486 и Pentium размер возможной виртуальной памяти может достигать 4 Тбайт.

Сегментированная модель памяти. *Сегментация* — механизм адресации, обеспечивающий существование нескольких независимых адресных пространств как в пределах одной задачи, так и в системе в целом для защиты задач от взаимного влияния. В основе механизма сегментации лежит понятие *сегмента*, который представляет собой независимый поддерживаемый на аппаратном уровне блок памяти.

Когда мы рассматривали сегментные регистры, то отмечали, что для процессоров Intel, начиная с i8086, принят особый подход к управлению памятью. Каждая программа в общем случае может состоять из любого количества сегментов, но непосредственный доступ она имеет только к трем основным сегментам (кода, данных и стека), а также к дополнительным сегментам данных числом от одного до трех. Программа никогда не знает, по каким физическим адресам будут размещены ее сегменты. Этим занимается операционная система. Операционная система размещает сегменты программы в оперативной памяти по определенным физическим адресам, после чего помещает значения этих адресов в определенные места. Куда именно, зависит от режима работы процессора. Так, в реальном режиме эти адреса помещаются непосредственно в соответствующие сегментные регистры, а в защищенном режиме они размещаются в элементы специальной системной дескрипторной таблицы. Внутри сегмента программа обращается к адресам относительно начала сегмента линейно, то есть начиная с 0 и заканчивая адресом, равным размеру сегмента. Этот относительный адрес, или *смещение*, который процессор использует для доступа к данным внутри сегмента, называется *эффективным*. Отличия моделей сегментированной организации памяти в различных режимах хорошо видны на схеме (рисунок 1.9). Различают три основных модели сегментированной организации памяти:

- сегментированная модель памяти реального режима;
- сегментированная модель памяти защищенного режима;
- сплошная модель памяти защищенного режима.

Рассмотрим порядок формирования физического адреса в реальном и

защищенном режимах. Уточним терминологию. Под физическим адресом понимается адрес памяти, выдаваемый на шину адреса процессора. Другое название этого адреса — линейный адрес. Подобная двойственность в названии обусловлена наличием страничной модели организации оперативной памяти. Эти названия являются синонимами только при отключении страничного преобразования адреса (в реальном режиме страничная адресация всегда отключена). Страничная модель, как мы отметили ранее, является надстройкой над сегментированной моделью. В страничной модели линейный и физический адреса имеют разные значения. Далее мы будем обсуждать схему, на которой показан порядок формирования адреса в реальном режиме работы процессора. Обратите внимание на наличие в этой схеме устройства страничного преобразования адреса, предназначенного для того, чтобы совместить две принципиально разные модели организации оперативной памяти и выдать на шину адреса истинное значение физического адреса памяти.

Формирование физического адреса в реальном режиме. Далее перечислены характеристики механизма адресации физической памяти в реальном режиме.

Диапазон изменения физического адреса — от 0 до 1 Мбайт. Эта величина определяется тем, что шина адреса i8086 имела 20 линий.

Максимальный размер сегмента — 64 Кбайт. Это объясняется 16-разрядной архитектурой i8086. Нетрудно подсчитать, что максимальное значение, которое могут содержать 16-разрядные регистры, составляет $2^{16} – 1$, что применительно к памяти и определяет величину 64 Кбайт.

Для обращения к конкретному физическому адресу оперативной памяти необходимо определить адрес начала сегмента (сегментную составляющую) и смещение внутри сегмента.

Понятие адреса начала сегмента ввиду принципиальной важности требует дополнительного пояснения. Исходя из разрядности сегментных регистров, можно утверждать, что сегментная составляющая адреса (или база сегмента) представляет собой всего лишь 16разрядное значение, помещенное в один из сегментных регистров. Максимальное значение, которое при этом получается, соответствует $2^{16} – 1$. Если так рассуждать, то получается, что адрес начала сегмента может быть только в диапазоне 0–64 Кбайт от начала оперативной памяти. Возникает вопрос, как адресовать остальную часть оперативной памяти вплоть до 1 Мбайт с учетом того, что размер самого сегмента не превышает 64 Кбайт.

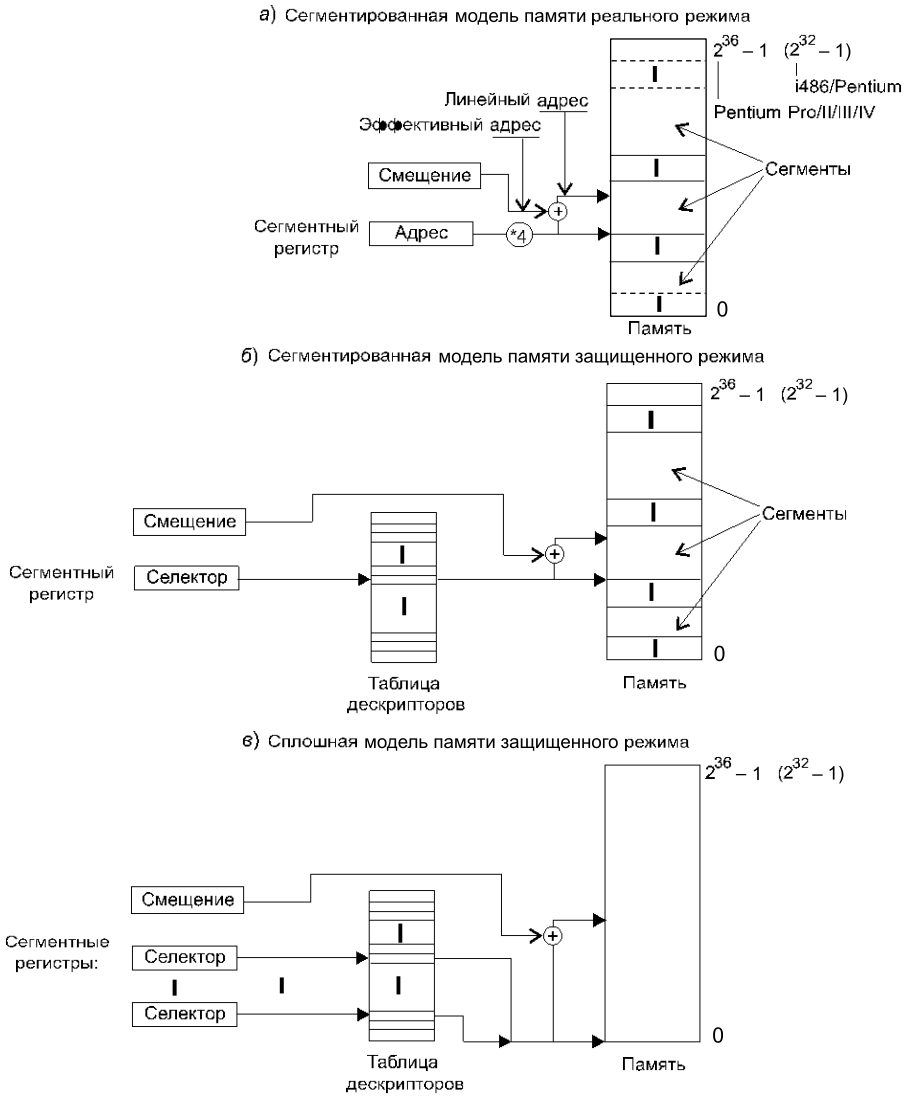


Рисунок 1.9 – Модели памяти процессоров Intel

Дело в том, что в сегментном регистре содержатся только старшие 16 битов физического адреса начала сегмента. Недостающие младшие четыре бита 20-разрядного адреса получаются сдвигом значения в сегментном регистре влево на 4 разряда. Эта операция сдвига выполняется аппаратно и для программного обеспечения абсолютно прозрачна. Получившееся 20-разрядное значение и является настоящим физическим адресом, соответствующим началу сегмента. Что касается второго компонента (смещения), участвующего в образовании физического адреса некоторого объекта в памяти, то он представляет собой 16-разрядное значение. Это значение может содержаться явно в команде либо косвенно в одном из регистров общего назначения. В процессоре эти две составляющие складываются на аппаратном уровне, в результате получается физический адрес памяти размерностью 20 битов. Данный механизм образования

физического адреса позволяет сделать программное обеспечение перемещаемым, то есть не зависящим от конкретных адресов загрузки его в оперативной памяти (рисунок 1.10).

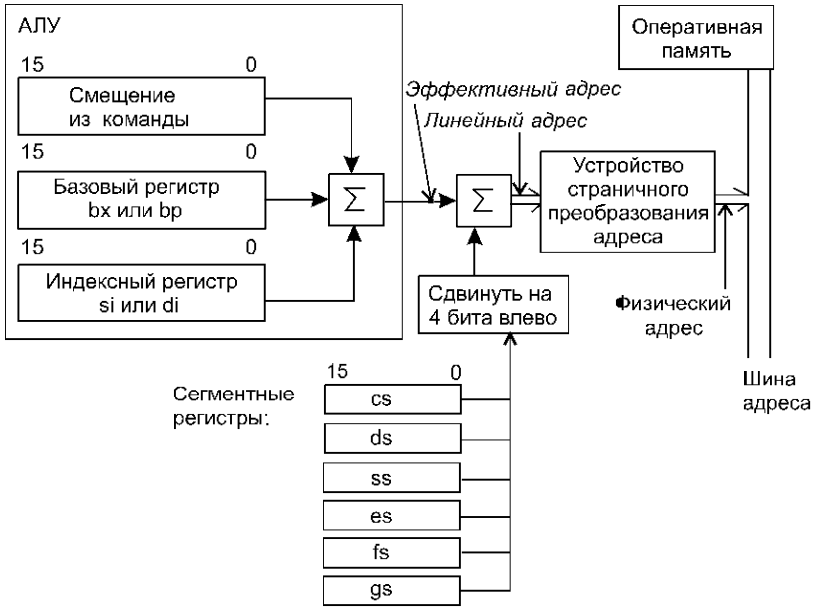


Рисунок 1.10 – Механизм формирования физического адреса в реальном режиме

На рисунке хорошо видно, как формируется некоторый целевой физический адрес: сегментная часть извлекается из одного из сегментных регистров, сдвигается на четыре разряда влево и суммируется со смещением. В свою очередь, видно, что значение смещения можно получить минимум из одного и максимум из трех источников: из значения смещения в самой машинной команде и/или из содержимого одного базового и/или одного индексного регистра. Количество источников, участвующих в формировании смещения, определяется кодированием конкретной машинной команды, и если таких источников несколько, то значения в них складываются. В заключение заметим, что не стоит волноваться из-за несоответствия размеров шины адреса процессора i486 или Pentium (32 бита) и 20 разрядного значения физического адреса реального режима. Пока процессор находится в реальном режиме, старшие 12 линий шины адреса попросту недоступны, хотя при определенных условиях и существует возможность работы с первыми 64 Кбайт оперативной памяти, лежащими сразу после первого мегабайта.

Недостатки такой организации памяти:

- сегменты бесконтрольно размещаются с любого адреса, кратного 16 (так как содержимое сегментного регистра аппаратно смещается на 4 разряда), и, как следствие, программа может обращаться по любым адресам, в том числе и реально не существующим;

- сегменты имеют максимальный размер 64 Кбайт;
- сегменты могут перекрываться другими сегментами.

Желанием ввести в архитектуру средства, позволяющие избавиться от указанных недостатков, и обусловлено, в частности, появление защищенного режима, в котором работают все современные операционные системы, в том числе Windows и Linux.

Формирование физического адреса в защищенном режиме. Основная идея защищенного режима — защитить исполняемые процессором программы от взаимного влияния. В защищенном режиме процессор поддерживает два типа защиты — по привилегиям и по доступу к памяти. В контексте нашего изложения интерес представляет второй тип защиты. Его мы и рассмотрим.

Для введения любого механизма защиты нужно иметь как можно больше информации об объектах защиты. Для процессора такими объектами являются исполняемые им программы. Организуя защиту программ по доступу к памяти, фирма Intel не стала нарушать принцип сегментации, свойственный ее процессорам. Так как каждая программа занимает один или несколько сегментов в памяти, то логично иметь больше информации об этих сегментах как об объектах, реально существующих в данный момент в вычислительной системе. Если каждому из сегментов присвоить определенные атрибуты, то часть функций по контролю за доступом к ним можно переложить на процессор. Что и было сделано. Любой сегмент памяти в защищенном режиме имеет следующие атрибуты:

- расположение сегмента в памяти;
- размер сегмента;
- уровень привилегий (определяет права данного сегмента относительно других сегментов);
- тип доступа (определяет назначение сегмента);
- некоторые другие.

В отличие от реального режима, в защищенном режиме программа уже не может запросто обратиться по любому физическому адресу памяти. Для этого она должна иметь определенные полномочия и удовлетворять ряду требований.

Ключевым объектом защищенного режима является специальная структура — дескриптор сегмента, который представляет собой 8байтовый дескриптор (краткое описание) непрерывной области памяти, содержащий перечисленные ранее атрибуты. На рисунке 1.11 приведена структура дескриптора сегмента.

Приведем назначение некоторых полей дескриптора сегмента:

- limit_1 и limit_2 — 20-разрядное поле, определяющее размер сегмента;
- base_1 и base_2 — 32-разрядное поле, определяющее значение

линейного адреса начала сегмента в памяти;

- AR — байт, поля которого определяют права доступа к сегменту; D— бит разрядности операндов и адресов;

- G — бит гранулярности.

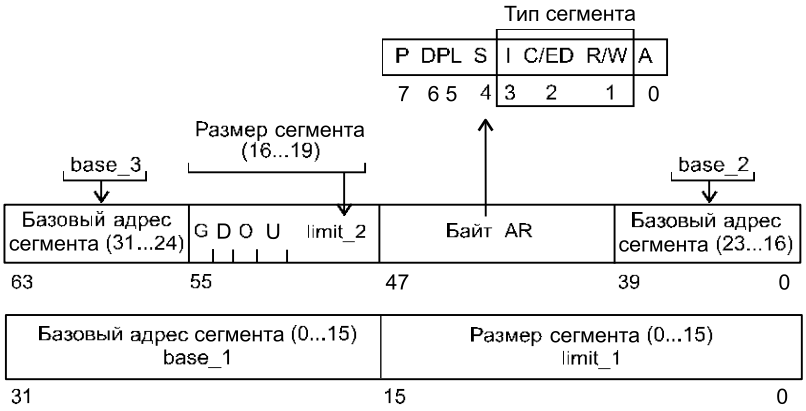


Рисунок 1.11 – Структура дескриптора сегмента защищенного режима процессора

В защищенном режиме размер сегмента не фиксирован, его расположение можно задать в пределах 4 Гбайт. Если посмотреть на рисунок, то возникнет вопрос: почему разорваны поля, определяющие размер сегмента и его начальный (базовый) адрес? Это результат эволюции процессоров. Защищенный режим впервые появился в процессоре i80286. Этот процессор имел 24-разрядную адресную шину и, соответственно, мог адресовать в защищенном режиме до 16 Мбайт оперативной памяти. Для этого ему достаточно было иметь в дескрипторе поле базового адреса 24 бита и поле размера сегмента 16 битов. После появления процессора i80386 с 32-разрядной шиной команд и данных в целях совместимости программ разработчики не стали менять формат дескриптора, а просто использовали свободные поля. Внутри процессора эти поля объединены. Внешне же они остались разделенными, и при программировании с этим приходится мириться.

Следующий интересный момент связан с тем, что размер сегмента в защищенном режиме может достигать 4 Гбайт, то есть занимать все доступное физическое пространство памяти. Как это возможно, если суммарный размер поля размера сегмента составляет всего 20 битов, что соответствует величине 1 Мбайт? Секрет скрыт в поле гранулярности — бит G (см. рисунок 1.11). Если G = 0, то значение в поле размера сегмента означает размер сегмента в байтах, если G = 1, — то в страницах. Размер страницы составляет 4 Кбайт. Нетрудно подсчитать, что когда максимальное значение поля размера сегмента составляет 0fffffh, то это

соответствует 1 М страниц или величине 1 М × 4 Кбайт = 4 Гбайт.

Выведение информации о базовом адресе сегмента и его размере на уровень процессора позволяет аппаратно контролировать работу программ с памятью и предотвращать обращения к несуществующим адресам либо к адресам, находящимся вне предела, разрешенного полем размера сегмента limit.

Другой аспект защиты заключается в том, что сегменты неравноправны в правах доступа к ним. Информация об этом содержится в специальном байте AR, входящем в состав дескриптора. Наиболее важные поля байта AR — это dpl и биты R/W, C/ED и I, которые вместе определяют тип сегмента. Поле dpl — часть механизма защиты по привилегиям. Суть этого механизма заключается в том, что конкретный сегмент может находиться на одном из четырех уровней привилегированности с номерами 0, 1, 2 и 3. Самым привилегированным является уровень 0. Существует ряд ограничений (опять-таки на аппаратном уровне) на взаимодействие сегментов кода, данных и стека с различными уровнями привилегий.

Итак, мы выяснили, что в защищенном режиме перед использованием любой области памяти должна быть проведена определенная работа по инициализации соответствующего дескриптора. Эту работу выполняет операционная система или программа, сегменты которой также описываются подобными дескрипторами. Все дескрипторы собираются вместе в одну из трех дескрипторных таблиц:

- глобальная дескрипторная таблица (Global Descriptor Table, GDT), ее адрес хранится в регистре GDTR;
- локальная дескрипторная таблица (Local Descriptor Table, LDT), ее адрес хранится в регистре LDTR;
- дескрипторная таблица векторов прерываний (Interrupt Descriptor Table, IDT), ее адрес хранится в регистре IDTR.

В какую именно таблицу должен быть помещен дескриптор, определяется его назначением. Адрес, по которому размещаются эти дескрипторные таблицы, может быть любым; он хранится в специально предназначенном для этого адреса системном регистре.

Схемы, показанные на рисунке 1.9, б и в, иллюстрируют принцип формирования адреса в защищенном режиме. Важно отметить изменение роли сегментных регистров. В защищенном режиме они содержат не адрес, а селектор, то есть указатель на соответствующую ячейку одной из дескрипторных таблиц (GDT или LDT).

Остальные элементы архитектуры IA-32, такие как формат машинных команд, типы данных и др., логично рассмотреть в следующих главах в контексте соответствующих аспектов использования языка ассемблера.

7 Организация ЭВМ и язык ассемблера

Ассемблер является машинно-ориентированным языком, символическим аналогом машинного языка. Это означает, что программист, пишущий на ассемблере, работает непосредственно с ресурсами компьютера, что требует хорошего понимания и знания его организации (архитектуры процессора, системного интерфейса, памяти, подсистемы ввода-вывода), логики работы операционной системы. С другой стороны, освоение и применение ассемблера является хорошим инструментарием для активного изучения логической и структурной организации компьютерных систем.

7.1 Общие сведения. Основные конструкции

Программа на ассемблере представляет собой совокупность блоков памяти, называемых сегментами. Программа может состоять из одного или нескольких таких блоков-сегментов.

Сегменты программы имеют определенное назначение, соответствующее типу сегментов: кода, данных и стека. Названия типов сегментов отражают их назначение. Деление программы на сегменты отражает сегментную организацию памяти процессоров Intel (архитектура IA-32). Каждый сегмент состоит из совокупности отдельных строк, в терминах теории компиляции называемых предложениями языка. Для языка ассемблера предложения, составляющие программу, могут представлять собой синтаксические конструкции четырех типов.

Команды(инструкции) представляют собой символические аналоги машинных команд. В процессе трансляции инструкции ассемблера преобразуются в соответствующие команды системы команд процессора.

Макрокоманды — это оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими предложениями.

Директивы являются указанием транслятору ассемблера на выполнение некоторых действий. У директив нет аналогов в машинном представлении.

Комментарии содержат любые символы, в том числе и буквы русского алфавита. Комментарии игнорируются транслятором.

Для распознавания транслятором ассемблера этих предложений их нужно формировать по определенным синтаксическим правилам.

На рисунках 1.12, 1.13 и 1.14 показан порядок написания предложений ассемблера с помощью синтаксических диаграмм.

Имя метки — символьный идентификатор. Значением данного идентификатора является адрес первого байта предложения программы, которому он предшествует.

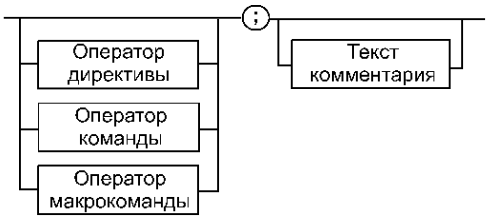


Рисунок 1.12 – Формат предложений ассемблера

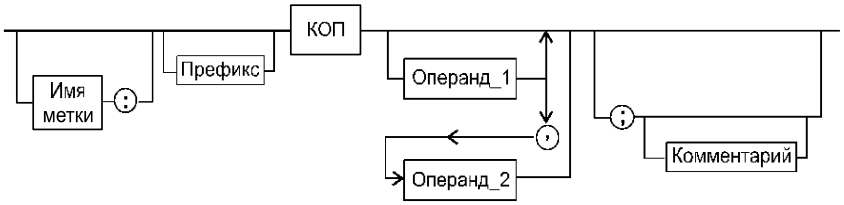


Рисунок 1.13 – Формат директив

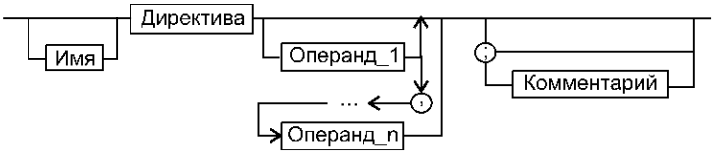


Рисунок 1.14 – Формат команд и макрокоманд

Префикс — символическое обозначение элемента машинной команды, предназначенного для изменения стандартного действия следующей за н

им команды ассемблера).

Имя — идентификатор, отличающий данную директиву от других одноименных директив. В зависимости от конкретной директивы в результате обработки ассемблером этому имени могут быть присвоены определенные характеристики.

Код операции (КОП) и директива — это мнемонические обозначения соответствующей машинной команды, макрокоманды или директивы транслятора.

Операнды. Операнды — это объекты, над которыми или при помощи которых выполняются действия, задаваемые инструкциями или директивами. Машинные команды могут либо совсем не иметь операндов, либо иметь один или два операнда. Большинство команд требует двух операндов, один из которых является источником, а другой — приемником (операндом назначения). В двухоперандной машинной команде возможны следующие сочетания операндов:

- *регистр — регистр;*
- *регистр — память;*
- *память — регистр;*
- *непосредственный операнд — регистр;*

- *непосредственный операнд — память.*

Здесь важно подчеркнуть, что один операнд может располагаться в регистре или памяти, а второй операнд обязательно должен находиться в регистре или непосредственно в команде. Непосредственный операнд может быть только источником.

Для приведенных ранее правил сочетания типов операндов есть исключения, которые касаются:

- команд работы с цепочками, которые могут перемещать данные из памяти в память;
- команд работы со стеком, которые могут переносить данные из памяти в стек, также находящийся в памяти;
- команд типа умножения, которые, кроме операнда, указанного в команде, неявно используют еще и второй операнд.

Операндами могут быть числа, регистры, ячейки памяти, символьные идентификаторы. При необходимости для расчета некоторого значения или определения ячейки памяти, на которую будет воздействовать данная команда или директива, используются выражения, то есть комбинации чисел, регистров, ячеек памяти, идентификаторов с арифметическими, логическими, побитовыми и атрибутивными операторами.

Директивы сегментации. В ходе предыдущего обсуждения были приведены основные правила записи команд и операндов в программе на ассемблере. Открытым остался вопрос о том, как правильно оформить последовательность команд, чтобы транслятор мог их обработать, а процессор — выполнить. В главах 2 и 3 мы уже касались понятия сегмента. При рассмотрении архитектуры процессора мы узнали, что он имеет шесть сегментных регистров, посредством которых может одновременно работать:

- с одним сегментом кода;
- с одним сегментом стека;
- с одним сегментом данных;
- с тремя дополнительными сегментами данных.

Еще раз вспомним, что физически сегмент представляет собой область памяти, занятую командами и/или данными, адреса которых вычисляются относительно значения в соответствующем сегментном регистре.

Синтаксическое описание сегмента на ассемблере представляет собой конструкцию, представленную на рисунке 1.15.

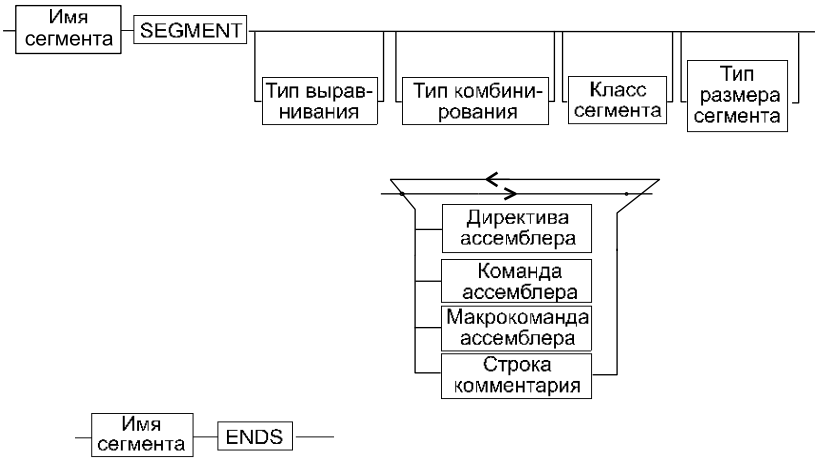


Рисунок 1.15 – Синтаксис описания сегмента

Важно отметить, что функциональное назначение сегментации несколько шире, чем простое разбиение программы на блоки кода, данных и стека. Сегментация является частью более общего механизма, связанного с концепцией модульного программирования. Она предполагает унификацию формата объектных модулей, создаваемых компилятором, в том числе компилируемых с разных языков программирования. Это позволяет объединять программы, написанные на разных языках. Именно для реализации различных вариантов такого объединения и предназначены операнды в директиве SEGMENT. Рассмотрим их подробнее.

Атрибут выравнивания сегмента (тип выравнивания) сообщает компоновщику о том, что нужно обеспечить размещение начала сегмента на заданной границе. Это важно, поскольку при правильном выравнивании доступ к данным в процессорах i80x86 выполняется быстрее. Допустимые значения этого атрибута приведены далее. По умолчанию тип выравнивания имеет значение PARA:

BYTE — выравнивание не выполняется. Сегмент может начинаться с любого адреса памяти;

WORD — сегмент начинается по адресу, кратному двум, то есть последний (младший) значащий бит физического адреса равен 0 (выравнивание по границе слова);

DWORD — сегмент начинается по адресу, кратному четырем, то есть два последних (младших) значащих бита равны 0 (выравнивание по границе двойного слова);

PARA — сегмент начинается по адресу, кратному 16, то есть последняя шестнадцатеричная цифра адреса должна быть 0h (выравнивание по границе параграфа);

PAGE — сегмент начинается по адресу, кратному 256, то есть две последние шестнадцатеричные цифры должны быть 00h (выравнивание по границе страницы размером 256 байт);

MEMPAGE — сегмент начинается по адресу, кратному 4 Кбайт, то есть три последние шестнадцатеричные цифры должны быть 000h (адрес следующей страницы памяти размером 4 Кбайт).

Атрибут комбинирования сегментов (комбинаторный тип) сообщает компоновщику, как нужно комбинировать сегменты различных модулей, имеющие одно и то же имя. По умолчанию атрибут комбинирования принимает значение PRIVATE. Возможные значения атрибута комбинирования сегмента перечислены далее:

PRIVATE — сегмент не будет объединяться с другими сегментами с тем же именем вне данного модуля;

PUBLIC — заставляет компоновщик объединить все сегменты с одинаковым именем. Новый объединенный сегмент будет целым и непрерывным. Все адреса (смещения) объектов, а это могут быть, в зависимости от типа сегмента, команды или данные, будут вычисляться относительно начала этого нового сегмента;

COMMON — располагает все сегменты с одним и тем же именем по одному адресу, то есть все сегменты с данным именем перекрываются. Размер полученного в результате сегмента будет равен размеру самого большого сегмента;

ATxxxx — располагает сегмент по абсолютному адресу параграфа (параграф — область памяти, объем которой кратен 16, потому последняя шестнадцатеричная цифра адреса параграфа равна 0). Абсолютный адрес параграфа задается выражением xxxx. Компоновщик располагает сегмент по заданному адресу памяти (это можно использовать, например, для доступа к видеопамяти или области ПЗУ), учитывая атрибут комбинирования. Физически это означает, что сегмент при загрузке в память будет расположен, начиная с этого абсолютного адреса параграфа, но для доступа к нему в соответствующий сегментный регистр должно быть загружено заданное в атрибуте значение. Все метки и адреса в определенном таким образом сегменте отсчитываются относительно заданного абсолютного адреса;

STACK — определение сегмента стека. Заставляет компоновщик объединить все одноименные сегменты и вычислять адреса в этих сегментах относительно регистра SS. Комбинированный тип STACK (стек) аналогичен комбинированному типу PUBLIC за исключением того, что регистр SS является стандартным сегментным регистром для сегментов стека. Регистр SP устанавливается на конец объединенного сегмента стека. Если не указано ни одного сегмента стека, компоновщик выдаст предупреждение, что стековый сегмент не найден. Если сегмент стека создан, а комбинированный тип STACK не используется, программист должен явно загрузить в регистр SS адрес сегмента (подобно

тому, как это делается для регистра DS).

Атрибут класса сегмента (тип класса) — это заключенная в кавычки строка, помогающая компоновщику определить нужный порядок следования сегментов при сборке программы из сегментов нескольких модулей. Компоновщик объединяет вместе в памяти все сегменты с одним и тем же именем класса (имя класса в общем случае может быть любым, но лучше, если оно отражает функциональное назначение сегмента). Типичным примером использования имени класса (обычно класса code) является объединение в группу всех сегментов кода программы. С помощью механизма типизации класса можно группировать также сегменты инициализированных и неинициализированных данных.

Атрибут размера сегмента. Для процессоров i80386 и выше сегменты могут быть 16- или 32-разрядными. Это влияет прежде всего на размер сегмента и порядок формирования физического адреса внутри него. Далее перечислены возможные значения атрибута:

USE16 — сегмент допускает 16-разрядную адресацию. При формировании физического адреса может использоваться только 16-разрядное смещение. Соответственно, такой сегмент может содержать до 64 Кбайт кода или данных;

USE32 — сегмент должен быть 32-разрядным. При формировании физического адреса может использоваться 32-разрядное смещение. Поэтому такой сегмент может содержать до 4 Гбайт кода или данных.

Все сегменты сами по себе равноправны, так как директивы SEGMENT и ENDS не содержат информации о функциональном назначении сегментов. Для того чтобы использовать их как сегменты кода, данных или стека, необходимо предварительно сообщить транслятору об этом с помощью специальной директивы ASSUME, формат которой показан на рисунке 1.16. Эта директива сообщает транслятору, какой сегмент к какому сегментному регистру привязан. В свою очередь, это позволяет транслятору корректно связывать символические имена, определенные в сегментах. Привязка сегментов к сегментным регистрам осуществляется с помощью операндов этой директивы, в которых имя_сегмента должно быть именем сегмента, определенным в исходном тексте программы директивой SEGMENT или ключевым словом NOTHING. Если в качестве операнда используется только ключевое слово NOTHING, то предшествующие назначения сегментных регистров аннулируются, причем сразу для всех шести сегментных регистров. Ключевое слово NOTHING можно также использовать вместо аргумента имя сегмента; в этом случае будет выборочно разрываться связь между сегментом с именем имя сегмента и соответствующим сегментным регистром.

Рассмотренные ранее директивы сегментации используются для оформления программы в трансляторах MASM и TASM. Поэтому их называют стандартными директивами сегментации.

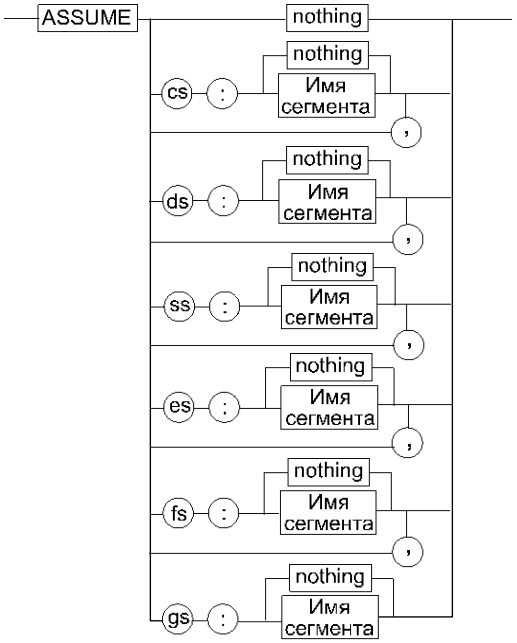


Рисунок 1.16 – Директива ASSUME

Для простых программ, содержащих по одному сегменту для кода, данных и стека, хотелось бы упростить их описание. Для этого в трансляторы MASM и TASM была введена возможность использования упрощенных директив сегментации. При этом возникла проблема, связанная с тем, что необходимо было как-то компенсировать невозможность напрямую управлять размещением и комбинированием сегментов. Для этого совместно с упрощенными директивами сегментации стали использовать директиву указания модели памяти MODEL, которая частично стала управлять размещением сегментов и выполнять функции директивы ASSUME (поэтому при введении в программу упрощенных директив сегментации директиву ASSUME можно не указывать). Директива MODEL связывает сегменты, которые при наличии упрощенных директив сегментации имеют предопределенные имена, с сегментными регистрами (хотя все равно придется явно инициализировать регистр DS).

Для сравнения приведем два листинга с программами на ассемблере. Функционально они одинаковы и выводят на консоль сообщение: «Hello World! No war and bomb! Let’s live friendly and learn assembler language». Листинг 1 содержит программу со стандартными директивами сегментации, а листинг 2, соответственно, — с упрощенными.

Листинг 1. Использование стандартных директив сегментации

```
data    segment para public 'data'
message db      'Hello World! No war and bomb! Let us live friendly and learn assembler
                language. $'
data    ends
stk     segment stack
```

```
        db 256 dup ('?'); сегмент стека
stk ends
code
segment para public 'code'      ; начало сегмента кода
main    proc    ; начало процедуры main
        assume cs:code,ds:data,ss:stk
        mov ax,data     ; адрес сегмента данных в регистр ax mov ds,ax    ; ax в ds
        mov ah,9
        mov dx,offset message
        int 21h  ; вывод сообщения на экран
        mov ax,4c00h  ; пересылка 4c00h в регистр ax
        int 21h  ; вызов прерывания с номером 21h
main    endp    ; конец процедуры main
code    ends    ; конец сегмента кода
end     main    ; конец программы с точкой входа main
```

Листинг 5.2. Использование упрощенных директив сегментации

```
masm    ; режим работы для TASM — masm, для MASM — не нужно
model    small  ; модель памяти
.data    ; сегмент данных
message db    'Hello World! No war and bomb! Let us live friendly and learn
                assembler language. $'
.stack   256h    ; сегмент стека .code ; сегмент кода
main    proc    ; начало процедуры main
        mov ax, @data; заносим адрес сегмента данных в регистр ax
        mov ds,ax       ; ax в ds
        mov ah,9
        mov dx,offset message
        int 21h  ; вывод сообщения на экран
        mov ax,4c00h  ; пересылка 4c00h в регистр ax
        int 21h  ; вызов прерывания с номером 21h
main    endp    ; конец процедуры main
end     main    ; конец программы с точкой входа main
```

Обязательным параметром директивы MODEL является модель_памяти. Этот параметр определяет модель сегментации памяти для программного модуля. Предполагается, что программный модуль может иметь только определенные типы сегментов, которые определяются так называемыми упрощенными директивами описания сегментов (таблица 1.2).

Упрощенные директивы определения сегментов режима masm:

.code [размер] – начало или продолжение сегмента кода;

.data – начало или продолжение сегмента инициализированных данных;

.const – начало или продолжение сегмента постоянных данных (констант);

.data? – начало или продолжение сегмента неинициализированных данных;

Таблица 1.2 – Модели памяти

Модель	Тип кода	Тип данных	Назначение модели
tiny	near	near	Код и данные объединены в одну группу с именем DGROUP. Используется для создания программ формата .com
small	near	near	Код занимает один сегмент, данные объединены в одну группу с именем DGROUP.
medium	far	near	Код занимает несколько сегментов, по одному на каждый объединяемый программный модуль. Все ссылки на передачу управления типа far. Данные объединены в одной группе, все ссылки на них – типа near
compact	near	far	Код в одном сегменте, ссылки на данные – типа far
large	far	far	Код в нескольких сегментах, по одному на каждый объединяемый программный модуль
flat	near	near	Код и данные в одном 32-битном сегменте (плоская модель памяти)

.stack [размер] – начало или продолжение сегмента стека;

.fardata [имя] – начало или продолжение сегмента инициализированных данных типа far;

.fardata? [имя] – начало или продолжение сегмента неинициализированных данных типа far.

Простые типы данных ассемблера. Любая программа предназначена для обработки некоторой информации, поэтому вопрос о том, какие типы данных языка программирования доступны для использования и какие средства языка привлекаются для их описания, обычно встает одним из первых. Трансляторы TASM и MASM предоставляют широкий набор средств описания и обработки данных, который вполне сравним с аналогичными средствами большинства языков высокого уровня.

Понятие типа данных носит двойственный характер. С точки зрения размерности процессор аппаратно поддерживает следующие основные типы данных (рис. 5.17).

Байт — восемь последовательно расположенных битов, пронумерованных от 0 до 7, при этом бит 0 является самым младшим значащим битом.

Слово — последовательность из двух байтов, имеющих последовательные адреса. Размер слова— 16битов;биты в слове нумеруются от 0 до 15. Байт, содержащий нулевой бит, называется младшим байтом, а байт, содержащий 15й бит, — старшим. Процессоры

Intel имеют важную особенность— младший байт всегда хранится по меньшему адресу. Адресом слова считается адрес его младшего байта. Адрес старшего байта может быть использован для доступа к старшей половине слова.

Двойное слово — последовательность из четырех байтов (32 бита), расположенных по последовательным адресам. Нумерация этих битов производится от 0 до 31. Слово, содержащее нулевой бит, называется младшим словом, а слово, содержащее 31й бит, — старшим словом. Младшее слово хранится по меньшему адресу. Адресом двойного слова считается адрес его младшего слова. Адрес старшего слова может быть использован для доступа к старшей половине двойного слова.

Учетверенное слово — последовательность из восьми байтов (64 бита), расположенных по последовательным адресам. Нумерация битов производится от 0 до 63. Двойное слово, содержащее нулевой бит, называется младшим двойным словом, а двойное слово, содержащее 63й бит, — старшим двойным словом. Младшее двойное слово хранится по меньшему адресу. Адресом учетверенного слова считается адрес его младшего двойного слова. Адрес старшего двойного слова может быть использован для доступа к старшей половине учетверенного слова.

128-битный упакованный тип данных. Этот тип данных появился в процессоре Pentium III. Для работы с ним в процессор введены специальные команды.

Кроме трактовки типов данных с точки зрения их разрядности, процессор на уровне команд поддерживает логическую интерпретацию этих типов.

Целый тип со знаком — двоичное значение со знаком размером 8, 16 или 32бита. Знак в этом двоичном числе содержится в 7, 15 или 31 бите соответственно. Ноль в этих битах в операндах соответствует положительному числу, а единица отрицательному. Отрицательные числа представляются в дополнительном коде. Числовые диапазоны для этого типа данных следующие:

- 8-разрядное целое — от –128 до +127;
- 16-разрядное целое — от –32 768 до +32 767; 32-разрядное целое — от –231 до +231 – 1.

Целый тип без знака — двоичное значение без знака размером 8, 16 или 32 бита. Числовой диапазон для этого типа следующий:

- байт — от 0 до 255;
- слово — от 0 до 65 535;
- двойное слово— от 0 до 232 – 1.

Указатель на память бывает двух типов:

- ближний тип — 32-разрядный логический адрес, представляющий собой относительное смещение в байтах от начала сегмента; указатели подобного типа могут также использоваться в сплошной (плоской) модели памяти, где сегментные составляющие одинаковы;

- дальний тип — 48-разрядный логический адрес, состоящий из двух частей: 16-разрядной сегментной части (селектора) и 32-разрядного смещения.

Цепочка представляет собой некоторый непрерывный набор байтов, слов или двойных слов максимальной длиной до 4 Гбайт.

Битовое поле представляет собой непрерывную последовательность битов, в которой каждый бит является независимым и может рассматриваться как отдельная переменная. Битовое поле может начинаться с любого бита любого байта и содержать до 32 битов.

Типы данных с плавающей точкой. Сопроцессор имеет несколько собственных типов данных, несовместимых с типами данных целочисленного устройства.

Типы данных MMX-расширения Pentium MMX/II/III/IV. Данный тип данных появился в процессоре Pentium MMX. Он представляет собой совокупность упакованных целочисленных элементов определенного размера.

Типы данных MMX-расширения Pentium III/IV. Этот тип данных появился в процессоре Pentium III. Он представляет собой совокупность упакованных элементов с плавающей точкой фиксированного размера.

Описанные ранее данные можно определить, как данные простого типа. Описать их можно с помощью специального вида директив — директив резервирования и инициализации данных. Эти директивы, по сути, являются указаниями транслятору на выделение определенного объема памяти. Если проводить аналогию с языками высокого уровня, то директивы резервирования и инициализации данных являются определениями переменных. Машинного эквивалента этим (впрочем, как и другим) директивам нет; просто транслятор, обрабатывая каждую такую директиву, выделяет необходимое количество байтов памяти и при необходимости инициализирует эту область некоторым значением. Формат директив резервирования и инициализации данных простых типов показан на рисунке 1.17.

На рисунке использованы следующие обозначения.

Знак вопроса (?) показывает, что содержимое поля не определено, то есть при задании директивы с таким значением выражения содержимое выделенного участка физической памяти изменяться не будет. Фактически, создается неинициализированная переменная.

Значение инициализации — значение элемента данных, которое будет занесено в память после загрузки программы. Фактически, создается инициализированная переменная, в качестве которой могут выступать константы, строки символов, константные и адресные выражения в зависимости от типа данных.

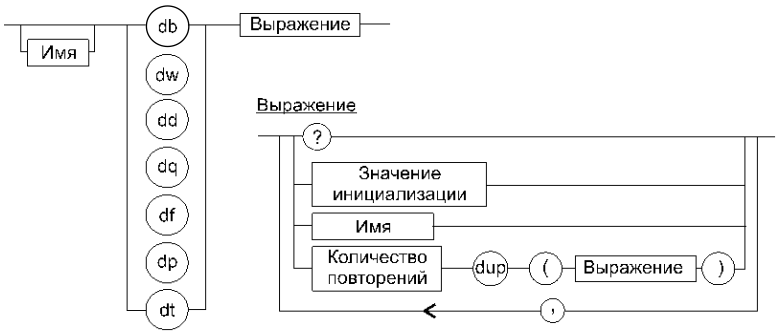


Рисунок 1.17 – Директивы описания данных простых типов

Выражение — итеративная конструкция, о синтаксисе которой можно судить по рисунку. В частности, она позволяет повторить занесение в физическую память выражения в скобках столько раз, сколько повторений указано.

Имя — некоторое символическое имя метки или ячейки памяти в сегменте данных, используемое в программе.

Далее представлены поддерживаемые TASM и MASM директивы резервирования и инициализации данных, а также информация о возможных типах и диапазонах значений, которые можно описывать или задавать с их помощью.

DB — резервирование памяти для данных размером 1 байт. Директивой DB можно задавать следующие значения:

- выражение или константу, принимающую значение из диапазона –128...+127 (для чисел со знаком) или 0...255 (для чисел без знака);
- 8-разрядное относительное выражение, использующее операции HIGH и LOW;
- символьную строку из одного или более символов, которая заключается в кавычки (в этом случае определяется столько байтов, сколько символов в строке).

DW — резервирование памяти для данных размером два байта. Директивой DW можно задавать следующие значения:

- выражение или константу, принимающую значение из диапазона –32 768...32 767 (для чисел со знаком) или 0...65 535 (для чисел без знака);
- выражение, занимающее 16 или менее битов, в качестве которого может выступать смещение в 16-битовом сегменте или адрес сегмента;
- 1- или 2-байтовая строка, заключенная в кавычки.

DD — резервирование памяти для данных размером четыре байта. Директивой DD можно задавать следующие значения:

- выражение или константу, принимающую значение из диапазона –32 768...+32 767 (для чисел со знаком и процессора i8086), 0...65 535 (для чисел без знака и процессора i8086), –2 147 483 648...+2 147 483 647 (для чисел со знаком и процессора i386 и выше) или 0...4 294 967 295 (для чисел без знака и процессора i386 и выше);
- относительное или адресное выражение, состоящее из 16-

разрядного адреса сегмента и 16-разрядного смещения;

- строку длиной до 4 символов, заключенную в кавычки.

DF и DP — резервирование памяти для данных размером 6 байтов.

DQ — резервирование памяти для данных размером 8 байтов.

DT — резервирование памяти для данных размером 10 байтов.

На линейных участках работают следующие группы:

- команды пересылки данных;
- арифметические команды
- логические команды;
- команды управления состоянием процессора.

7.2 Команды пересылки данных

К группе команд пересылки данных относятся следующие команды:

```
mov <операнд назначения>, <операнд-источник>
xchg <операнд1>,<операнд2>
```

mov — это основная команда пересылки данных. Она реализует самые разнообразные варианты пересылки. Отметим особенности применения этой команды.

Командой mov нельзя осуществить пересылку из одной области памяти в другую. Если такая необходимость возникает, то нужно использовать в качестве промежуточного буфера любой доступный в данный момент регистр общего назначения.

Нельзя загрузить в сегментный регистр значение непосредственно из памяти. Для такой загрузки требуется промежуточный объект. Это может быть регистр общего назначения или стек. Если вы посмотрите на листинг 5.1, то увидите в начале сегмента кода две команды mov, выполняющие настройку сегментного регистра ds. При этом из-за невозможности напрямую загрузить в сегментный регистр значение адреса сегмента, содержащееся в предопределенной переменной @data, приходится использовать регистр общего назначения ax.

Нельзя переслать содержимое одного сегментного регистра в другой сегментный регистр. Это объясняется тем, что в системе команд нет соответствующего кода операции. Но необходимость в таком действии часто возникает. Выполнить такую пересылку можно, используя в качестве промежуточных все те же регистры общего назначения. Вот пример инициализации регистра es значением из регистра ds:

```
mov ax, ds
mov es, ax
```

Но есть и другой, более «красивый» способ выполнения данной операции — использование стека и команд push и pop:

```
push ds        ;поместить значение регистра ds в стек
pop es         ;записать в es число из стека
```

Нельзя использовать сегментный регистр cs в качестве операнда назначения. Дело в том, что в архитектуре процессора IA-32 пара cs:ip содержит адрес команды, которая должна выполняться следующей. Изменение командой mov содержимого регистра cs фактически означало бы операцию перехода, а не пересылки, что недопустимо.

Для двунаправленной пересылки данных применяют команду xchg. Для этой операции можно, конечно, применить последовательность из нескольких команд mov, но из-за того что операция обмена используется довольно часто, разработчики системы команд процессора посчитали нужным ввести отдельную команду об-мена — xchg. Естественно, что операнды должны иметь один тип. Не допускается (как и для всех команд ассемблера) напрямую обменивать между собой содержимое двух ячеек памяти. К примеру,

```
xchg ax,bx                  ; обменять содержимое регистров ax и bx
xchg ax,word ptr [si]       ; обменять содержимое регистра ax и слова в
                            ; памяти по адресу в [si]
```

7.3 Работа с адресами и указателями

При написании программ на ассемблере производится интенсивная работа с адресами операндов, находящимися в памяти. Для поддержки такого рода операций есть специальная группа команд, в которую входят следующие команды:

lea <приемник>,<источник> — загрузка эффективного адреса;
lds <приемник>,<источник> — загрузка указателя в регистр сегмента данных ds;
les <приемник>,<источник> — загрузка указателя в регистр дополнительного сегмента данных es;
lgs <приемник>,<источник> — загрузка указателя в регистр дополнительного сегмента данных gs;
lfs <приемник>,<источник> — загрузка указателя в регистр дополнительного сегмента данных fs;
lss <приемник>,<источник> — загрузка указателя в регистр сегмента стека ss.

Команда lea похожа на команду mov тем, что она также производит пересылку, однако команда lea производит пересылку не данных, а эффективного адреса данных (то есть смещения данных относительно начала сегмента данных) в регистр, указанный операндом <приемник>.

7.4 Работа со стеком

Стек — это область памяти, специально выделяемая для временного хранения данных программы. Важность стека определяется тем, что для него в структуре программы предусмотрен отдельный сегмент. На тот случай, если программист забыл описать сегмент стека в своей программе, компоновщик tlink выдаст предупреждающее сообщение.

Для работы со стеком предназначены три регистра:

- ss — регистр сегмента стека;
- sp/esp — регистр указателя стека;
- bp/ebp — регистр указателя базы кадра стека.

Размер стека зависит от режима работы процессора и ограничивается значением 64 Кбайт (или 4 Гбайт в защищенном режиме). В каждый момент времени доступен только один стек, адрес сегмента которого содержится в регистре ss. Этот стек называется текущим. Для того чтобы обратиться к другому стеку («переключить стек»), необходимо загрузить в регистр ss другой адрес. Регистр ss автоматически используется процессором для выполнения всех команд, работающих со стеком.

Перечислим еще некоторые особенности работы со стеком.

Запись и чтение данных в стеке осуществляются в соответствии с принципом LIFO (Last In First Out — «последним пришел, первым ушел»).

По мере записи данных в стек последний растет в сторону младших адресов. Эта особенность заложена в алгоритм команд работы со стеком.

При использовании регистров esp/sp и ebp/bp для адресации памяти ассемблер автоматически считает, что содержащиеся в нем значения представляют собой смещения относительно сегментного регистра ss.

Регистры ss, esp/sp и ebp/bp используются комплексно, и каждый из них имеет свое функциональное назначение. Регистр esp/sp всегда указывает на вершину стека, то есть содержит смещение, по которому в стек был занесен последний элемент. Команды работы со стеком неявно изменяют этот регистр так, чтобы он указывал всегда на последний записанный в стек элемент. Если стек пуст, то значение esp равно адресу последнего байта сегмента, выделенного под стек. При занесении элемента в стек процессор уменьшает значение регистра esp, а затем записывает элемент по адресу новой вершины. При извлечении данных из стека процессор копирует элемент, расположенный по адресу вершины, а затем увеличивает значение регистра указателя стека esp. Таким образом, получается, что стек растет вниз, в сторону уменьшения адресов.

Что нужно сделать для получения доступа к элементам не на вершине, а внутри стека? Для этого применяют регистр ebp. Регистр ebp — регистр указателя базы кадра стека. Например, типичным приемом при входе в подпрограмму является передача нужных параметров путем записи их в стек. Если подпрограмма тоже активно работает со стеком, то доступ к этим параметрам становится проблематичным. Выход в том, чтобы после записи нужных данных в стек сохранить адрес вершины стека в указателе базы кадра стека — регистре ebp. Значение в ebp в дальнейшем можно использовать для доступа к переданным параметрам.

Для организации работы со стеком существуют специальные команды записи и чтения.

Команда push выполняет запись значения <источник> в вершину стека:

```
push <источник>
```

Интерес представляет алгоритм работы этой команды, который включает два действия:

1. Значение sp уменьшается на 2: (sp) = (sp) – 2.
2. Значение источника записывается по адресу, указываемому парой ss:sp.

Команда pop выполняет запись значения из вершины стека по месту, указанному операндом <приемник> (значение при этом «снимается» с вершины стека):

```
pop <приемник>
```

Алгоритм работы команды pop обратен алгоритму команды PUSH.

1. Запись содержимого вершины стека по месту, указанному операндом <приемник>.
2. Увеличение значения sp: (sp) = (sp) + 2

Команда pusha предназначена для групповой записи в стек. По этой команде в стек последовательно записывается содержимое регистров ax, cx, dx, bx, sp, bp, si, di. Заметим, что записывается оригинальное содержимое sp, то есть то, которое было до выдачи команды pusha.

Следующая действия, обратные действиям описанной ранее команды:

```
popa
```

Представленная далее группа команд позволяет сохранить в стеке регистр флагов и записать слово или двойное слово. Отметим, что перечисленные команды — единственные в системе команд процессора, которые позволяют получить доступ (и которые нуждаются в этом доступе) ко всему содержимому регистра флагов.

Команда pushf сохраняет регистр флагов в стеке. Работа этой команды зависит от атрибута размера сегмента:

- use16 — в стек записывается регистр flags размером два байта;
- use32 — в стек записывается регистр eflags размером четыре байта.

Команда PUSHFW сохраняет в стеке регистр флагов размером в слово. С атрибутом use16 всегда работает так же, как команда PUSHF.

Команда pushfd сохраняет в стеке регистр флагов flags или eflags в зависимости от атрибута размера сегмента (то есть то же, что и pushf).

Следующие три команды выполняют действия, обратные действиям рассмотренных выше команд:

```
popf;
popfw;
popfd.
```

Работать со стеком приходится постоянно, поэтому к этому вопросу мы будем возвращаться еще не раз. Отметим основные виды операций, когда использование стека практически неизбежно:

- вызов подпрограмм;
- временное сохранение значений регистров;
- определение локальных переменных в процедуре.

7.5 Арифметические команды

7.5.1 Команды сложения-вычитания

Команды сложения. Формат команды:

```
add <приемник>, <источник>;
```

Команда add используется для сложения приемника и источника, результат помещается в приемник.

Оба операнда не могут быть именами переменных. Источником может быть число.

Можно складывать: память+регистр, регистр+память, регистр+регистр, память+число, регистр+число. Нельзя складывать: память+память.

```
inc <приемник>;
```

Команда inc используется для увеличения содержимого приемника на единицу. Приемник – это регистр или ячейка памяти.

Команды вычитания. Формат команды:

```
sub <приемник>, <источник>;
```

Команда sub используется для вычитания содержимого источника из содержимого приемника, результат помещается в приемник.

Оба операнда не могут быть именами переменных. Источником может быть число.

```
dec <приемник>;
```

Команда dec используется для уменьшения содержимого приемника на единицу. Приемник – это регистр или ячейка памяти.

7.5.2 Команды умножения-деления

Команды умножения. Формат команды умножения чисел без знака:

```
mul <источник (И)> ;al*И→ax          при работе с байтами
                   ;ax*И→ax-dx       при работе со словами
```

Формат команды умножения чисел со знаком:

```
imul <источник (И)> ;al*И→ax         при работе с байтами
                    ;ax*И→ax-dx      при работе со словами
```

Команды mul и imul умножают содержимое регистра al или ax (в зависимости от размерности операндов) на содержимое источника, указанного в команде умножения. Источник – это или регистр или ячейка памяти. В результате работы команд умножения с данными длиной байт регистр al расширяется до ax, а с данными длиной слово регистр ax расширяется до dx.

Нельзя использовать в команде умножения в качестве источника непосредственное значение.

Команды деления. Формат команды деления чисел без знака:

```
div <источник (И)>  ;al/И→al-ah       при работе с байтами
                    ;ax/И→ax-dx       при работе со словами
                    ;al – частное, ah – остаток
                    ;ax – частное, dx – остаток
```

Формат команды деления чисел со знаком:

```
idiv <источник (И)>  ;al/И→al-ah          при работе с байтами
                     ;ax/И→ax-dx          при работе со словами
                     ;al – частное, ah    – остаток
                     ;ax – частное, dx    – остаток
```

Команды div и idiv делят содержимое регистра al или ax (в зависимости от размерности операндов) на содержимое источника, указанного в команде умножения. Источник – это или регистр или ячейка памяти. В результате работы команд умножения с данными длиной байт в регистр al помещается частное, в регистр ah – остаток, а с данными длиной слово – в регистр ax помещается частное, в регистр dx – остаток.

Нельзя использовать в команде деления в качестве источника непосредственное значение.

7.5.3 Команды расширения знака

Форматы команд:

```
cbw     ;расширить байт до слова
cwd     ;расширить слово до двойного слова
```

Команда cbw воспроизводит 7 бит регистра al во всех битах регистра ah.

Команда cwd воспроизводит 15 бит регистра ax во всех битах регистра dx.

Пример: вычислить значение выражения y=(7(a+7b)–3)/4.

```
;Программа работы с байтами
dseg    segment para public 'data'
        a db 3
        b db 2
        y db ?
 mes db 'конец программы$'
 dseg   ends
 sseg   segment para stack 'stack'
        db 30 dup (0)
 sseg   ends
 cseg   segment para public 'code'
        osn     proc near
                assume cs:cseg,ds:dseg,ss:sseg
                mov ax,dseg
                mov ds,ax
                mov bl,4        ;bl=4
```

```
                mov al,7        ;al=7
                imul b          ;al=7*b
                add al,a        ;al=a+7*b
                mov cl,7        ;cl=7
                imul cl         ;al=7*(a+7*b)
                sub al,3        ;al=7*(a+7*b)-3 cbw
                idiv bl         ;al=(7*(a+7*b)-3)/4 остаток в ah
                mov y,al        ;y=al
                lea dx,mes      ;вывод сообщения 'конец программы'
                mov ax,0900H
                int 21H
                mov ax,4C00H        ;завершение программы
                int 21H
        osn     endp
cseg ends
end osn
```

7.6 Программирование ветвящихся вычислительных процессов

Для программирования ветвящихся вычислительных процессов, используются команда вычитания cmp и команды передачи управления.

Команда вычитания. Формат команды вычитания:

```
cmp <приемник>, <источник>
```

Эта команда вычитает операнд-источник из операнда-приемника, но не сохраняет результат вычитания в операнде-приемнике, а только соответствующим образом воздействует на флаги.

Команды передачи управления. Команды передачи управления делятся на 4 группы:

1. Команды безусловной передачи управления.
2. Команды условной передачи управления.
3. Команды управления циклами.
4. Команды работы с процедурами.

Команды условной передачи управления. Команды условной передачи управления позволяют принять решение в зависимости от определенного условия. Если условие истинно, то осуществляется переход по указанной в команде метке. В противном случае выполняется команда, следующая за командой перехода.

Обычно команды условной передачи управления используются

совместно с командой сравнения cmp.

Формат команды условного перехода:

jx <метка>,

где x – модификатор команды;

<метка> – метка перехода, которая находится не далее –128 или +127 байтов от команды условной передачи.

В таблице 1.3. представлены некоторые команды условного перехода.

Таблица 1.3 – Команды условного перехода

Условие перехода	Следующая за cmp команда	
	для чисел без знака	для чисел со знаком
приемник > источник	ja	jg
приемник = источник	je	je
приемник < источник	jb	jl
приемник ≥ источник	jae	jge
приемник ≤ источник	jbe	jle
приемник ≠ источник	jne	jne

Пример:

```
        …
        cmp ax, bx   ;сравниваем содержимое регистров ax и bx
        jg met1      ;если ax>bx то переход на met1
        jl met2      ;если ax<bx то переход на met2
        je met3      ;если ax=bx то переход на met3
        …
met1:
        …
met2:
        …
met3:
        …
```

Команда безусловного перехода. Эта команда используется для обхода группы команд, которым передается управление из другой части программы.

Формат команды:

jmp <метка>,

где метка – имя метки перехода, которая находится не далее –128 или

+127 байтов от команды безусловного перехода.

Псевдооператоры определения процедур. Формат описания процедуры:

```
<имя_процедуры> proc <атрибут_дистанции>
...
ret
<имя_процедуры> endp
```

Каждая процедура должна начинаться оператором proc и заканчиваться оператором endp. Процедура должна содержать команду возврата из процедуры ret.

Атрибуты дистанции:
- far – дальняя процедура;
- near – близкая процедура.

Процедура с атрибутом near может быть вызвана только из того сегмента команд, в котором она определена.

Формат вызова процедуры:

```
call <имя_процедуры>
```

7.7 Программирование циклических вычислительных процессов

Команды управления циклами.

Команда loop.

Эта команда уменьшает содержимое регистра cx на 1 и передает управление оператору, помеченному меткой, если содержимое регистра cx≠0. Завершение выполнения цикла происходит в том случае, если содержимое регистра cx уменьшается до нуля.

Формат команды:

```
loop <метка>
```

Пример:

```
        …
        mov cx,10
met1:
        …
        ;тело цикла
        loop met1
```

Команда loope (loopz).

Эта команда уменьшает содержимое регистра cx на 1, а затем осуществляет переход, если содержимое регистра cx≠0 или флаг нуля zf=1. Повторение цикла завершается, если либо содержимое регистра cx=0, либо флаг zf=0, либо оба они равны 0. Команда loope обычно используется для поиска первого ненулевого результата в серии операций. Синонимом команды loope является команда loopz.

Команда loopne (loopnz).

Эта команда уменьшает содержимое регистра cx на 1, а затем осуществляет переход, если содержимое регистра cx≠0 и флаг нуля zf=0. Повторение цикла завершается, если либо содержимое регистра cx=0, либо флаг zf=1, либо будет выполнено и то и другое. Команда loopne обычно используется для поиска первого нулевого результата в серии операций. Синонимом команды loopne является команда loopnz.

7.8 Режимы адресации

Под режимами адресации понимаются способы доступа к данным.

Все режимы адресации можно условно разделить на 7 групп.

Для доступа к операнду используется 20-битовый физический адрес.

Физический адрес операнда получается сложением значения смещения адреса операнда с содержимым сегментного регистра, предварительно дополненного четырьмя нулями.

Смещение адреса операнда называется исполнительным адресом. Исполнительный адрес показывает, на каком расстоянии в байтах располагается операнд от начала сегмента, в котором он находится.

Будучи 16-битовым числом без знака, исполнительный адрес позволяет получить доступ к операндам, находящимся выше начала сегмента на расстоянии 64 Кбайт.

При работе с адресацией операндов необходимо помнить, что микропроцессор хранит 16-битовые числа в обратном порядке, а именно: младшие биты числа в байте с меньшим адресом.

Регистровая. Этот режим предполагает, что микропроцессор извлекает операнд из регистра или загружает его в регистр.

```
mov cx, ax
inc di
```

Непосредственная. Этот режим адресации позволяет указывать 8- или 16-битовые значения константы в операнде-источнике.

```
mov cx, 100
```

```
mov al, -10
```

Непосредственный операнд может быть идентификатором, определенным операторами equ или =

```
k equ 100
…
mov cx,k
```

Прямая. При прямой адресации исполнительный адрес является составной частью команды. Микропроцессор добавляет этот исполнительный адрес к сдвинутому содержимому регистра данных ds и получает 20-битовый физический адрес. По этому адресу и находится операнд. Обычно прямая адресация применяется, если операндом служит метка переменной.

```
mov ax, table
```

Косвенная регистровая адресация. В этом случае исполнительный адрес операнда содержится в базовом регистре bx, регистре указателя базы bp, регистре sp или индексных регистрах si и di.

```
mov ax, [bx]
```

Смещение адреса в регистр можно поместить оператором offset или lea

```
mov bx, offset table
lea bx, table
```

Адресация по базе. Ассемблер вычисляет исполнительный адрес с помощью сложения значения сдвига с содержимым регистров bx или bp. Адресация по базе используется при доступе к структурированным записям данных, расположенных в разных участках памяти. В этом случае базовый адрес записи помещается в базовый регистр bx или bp и доступ к отдельных его элементам записи осуществляется по сдвигу относительно базы. А для доступа к разным записям одной и той же структуры достаточно соответствующим образом изменить содержимое базового регистра.

```
mas db 1,2,3,4,5,6,7,8,9,10
mov bx, offset mas
mov al,[bx+4] ;загрузка в al mas[4]=5
```

Прямая адресация с индексированием. Исполнительный адрес

вычисляется как сумма значений смещения и индексного регистра di или si. Этот тип адресации удобен для доступа к элементам таблицы, когда смещение указывает на начало таблицы, а индексный регистр на номер элемента.

```
table db 10 dup (?)
mov di, 5
mov al, table[di] ;загрузка 6 элемента таблицы
```

Адресация по базе с индексированием. Исполнительный адрес вычисляется как сумма значений базового регистра, индексного регистра и сдвига. Этот режим адресации удобен при адресации двумерного массива. Пусть задан двумерный массив A:

$$A=\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

```
;выделение памяти под элементы матрицы
a db 1,2,3,4,5,6,7,8,9,10,11,12
;доступ к элементу a[1,2]=7 двумерного массива
mov bx, offset a
mov di,4
mov al, [bx][di+2]
```

Допускаются следующие формы записи адресации по базе с индексированием:

```
mov al, [bx+2+di]
mov al, [di+bx+2]
mov al, [bx+2][di]
```

Из семи режимов адресации самыми быстрыми являются регистровая и непосредственная адресации операндов. В других режимах адресация выполняется дольше, так как вначале необходимо вычислить адрес ячейки памяти, извлечь операнд, а затем передать его операционному блоку.

7.9 Цепочечные команды

Цепочечные команды также называют командами обработки строк символов. Под строкой символов понимается последовательность байт, а цепочка – это более общее название для случаев, когда элементы последовательности имеют размер слово или двойное слово. То есть цепочечные команды позволяют проводить действия над блоками памяти,

представляющими собой последовательности элементов следующего размера: 8 бит (байт); 16 бит (слово); 32 бита (двойное слово). Содержимое этих блоков для микропроцессора не имеет никакого значения. Это могут быть символы, числа и все что угодно. Главное, чтобы размерность элементов совпадала с одной из вышеперечисленных, и эти элементы находились в соседних ячейках памяти.

В системе команд микропроцессора имеются семь операций-примитивов обработки цепочек. Каждая из них реализуется в микропроцессоре тремя командами, в свою очередь, каждая из этих команд работает с соответствующим размером элемента – байтом, словом или двойным словом. Особенность всех цепочечных команд в том, что они, кроме обработки текущего элемента цепочки, осуществляют еще и автоматическое продвижение к следующему элементу данной цепочки.

Операции-примитивы и команды, с помощью которых они реализуются:

1. Пересылка цепочки:

```
movs <адрес_приемника>, <адрес_источника>
                ;(MOVe String) – переслать цепочку;
movsb           ;(MOVe String Byte) – переслать цепочку байтов;
movsw           ;(MOVe String Word) – переслать цепочку слов;
movsd           ;(MOVe String Double word) – переслать цепочку двойных
                ;слов.
```

Команды производят копирование элементов из одной области памяти (цепочки) в другую.

2. Сравнение цепочек:

```
cmps <адрес_приемника>, <адрес_источника>
                ;(CoMPare String) – сравнить строки;
cmpsb           ;(CoMPare String Byte) – сравнить строку байт;
cmpsw           ;(CoMPare String Word) – сравнить строку слов;
cmpsd           ;(CeMPare String Double word) – сравнить строку двойных
                ;слов.
```

Команды производят сравнение элементов цепочки-источника с элементами цепочки-приемника.

3. Сканирование цепочки:

```
scas <адрес_приемника> ;(SCAning String) – сканировать цепочку;
scasb           ;(SCAning String Byte) – сканировать цепочку байт;
scasw           ;(SCAning String Word) – сканировать цепочку слов;
scasd           ;(SCAning String Double Word) – сканировать цепочку
                ;двойных слов.
```

Команды производят поиск некоторого значения в области памяти.

4. Загрузка элемента из цепочки:

```
lods <адрес_источника> ;(LOaD String) – загрузить элемент из цепочки
                       ;в регистр-аккумулятор al/ax/eax;
lodsb                  ;(LOaD String Byte) – загрузить байт из цепочки в регистр al;
lodsw                  ;(LOaD String Word) – загрузить слово из цепочки в регистр ax;
lodsd                  ;(LOaD String Double Word) – загрузить двойное слово
                       ;из цепочки в регистр eax.
```

Эта операция позволяет извлечь элемент цепочки и поместить его в регистр-аккумулятор al, ax или eax.

5. Сохранение элемента в цепочке:

```
stos <адрес_приемника> ;(STOre String) – сохранить элемент из
                       ;регистра-аккумулятора al/ax/eax в цепочке;
stosb                  ;(STOre String Byte) – сохранить байт из регистра al в цепочке;
stosw                  ;(STOre String Word) – сохранить слово из регистра ax
                       ;в цепочке;
stosd                  ;(STOre String Double Word) – сохранить двойное слово из
                       ;регистра eax в цепочке.
```

Эта операция позволяет произвести действие, обратное команде lods, то есть сохранить значение из регистра-аккумулятора в элементе цепочки.

6. Получение элементов цепочки из порта ввода-вывода:

```
ins <адрес_приемника>, <номер_порта> ;(INput String) – ввести элементы из
                       ;порта ввода-вывода в цепочку;
insb                   ;(INput String Byte) – ввести из порта цепочку байтов;
insw                   ;(INput String Word) – ввести из порта цепочку слов;
insd                   ;(INput String Double Word) – ввести из порта цепочку двойных
                       ;слов.
```

7. Вывод элементов цепочки в порт ввода-вывода:

```
outs <номер_порта>, <адрес_источника>     ;(OUTput String) – вывести
                       ; элементы из цепочки в порт ввода-вывода;
outsb                  ;(OUTput String Byte) – вывести цепочку байтов в порт
                       ;ввода-вывода;
outsw                  ;(OUTtput String Word) – вывести цепочку слов в порт
                       ;ввода-вывода;
outsd                  ;(OUTput String Double Word) – вывести цепочку двойных слов
                       ;в порт ввода-вывода.
```

К цепочечным командам нужно отнести и *префиксы повторения:*

rep
repe или repz
repne или repnz

Префиксы повторения указываются перед нужной цепочечной командой в поле метки. Цепочечная команда без префикса выполняется один раз. Размещение префикса перед цепочечной командой заставляет ее выполняться в цикле. Отличия префиксов в том, на каком основании принимается решение о циклическом выполнении цепочечной команды: по состоянию регистра ecx/cx или по флагу нуля zf:

- префикс повторения rep (REPeat) используется с командами, реализующими операции-примитивы пересылки и сохранения элементов цепочек movs и stos. Префикс rep заставляет данные команды выполняться, пока содержимое в ecx/cx не станет равным 0. При этом цепочечная команда, перед которой стоит префикс, автоматически уменьшает содержимое ecx/cx на единицу. Та же команда, но без префикса, этого не делает;

- префиксы повторения repe или repz (REPeat while Equal or Zero) являются синонимами. Цепочечная команда выполняется до тех пор, пока содержимое ecx/cx не равно нулю или флаг zf равен 1. Как только одно из этих условий нарушается, управление передается следующей команде программы. Благодаря возможности анализа флага zf наиболее эффективно эти префиксы можно использовать с командами cmps и scas для поиска отличающихся элементов цепочек;

- префиксы повторения repne или repnz (REPeat while Not Equal or Zero) также являются синонимами. Префиксы repne/repnz заставляют цепочечную команду циклически выполняться до тех пор, пока содержимое ecx/cx не равно нулю или флаг zf равен нулю. При невыполнении одного из этих условий работа команды прекращается. Данные префиксы также можно использовать с командами cmps и scas, но для поиска совпадающих элементов цепочек.

Особенность формирования физического адреса операндов адрес_источника и адрес_приемника. Цепочка-источник, адресуемая операндом адрес_источника, может находиться в текущем сегменте данных, определяемом регистром ds. Цепочка-приемник, адресуемая операндом адрес_приемника, должна быть в дополнительном сегменте данных, адресуемом сегментным регистром es. Допускается замена (с помощью префикса замены сегмента) только регистра ds, регистр es заменять нельзя. Вторые части адресов – смещения цепочек должны находиться:

- для цепочки-источника это регистр esi/si;

- для цепочки-получателя это регистр edi/di.

Таким образом, полные физические адреса для операндов цепочечных команд следующие:

- адрес_источника – пара ds:esi/si;
- адрес_приемника – пара es:edi/di.

Есть две возможности задания направления обработки цепочки: от начала цепочки к её концу, то сеть в направлении возрастания адре- сов; от конца цепочки к началу, то есть в направлении убывания адресов.

Направление определяется значением флага направления df (Direction Flag) в регистре eflags/flags:

- если df=0, то значения индексных регистров esi/si и edi/di будут автоматически увеличиваться (операция инкремента) цепочечными командами, то есть обработка будет осуществляться в направлении возрастания адресов;
- если df=1, то значения индексных регистров esi/si и edi/di будут автоматически уменьшаться (операция декремента) цепочечными командами, то есть обработка будет идти в направлении убывания адресов.

Состоянием флага df можно управлять с помощью двух команд, не имеющих операндов:

- cld (Clear Direction Flag) – очистить флаг направления. Команда сбрасывает флаг направления df в 0.
- std (Set Direction Flag) – установить флаг направления. Команда устанавливает флаг направления df в 1.

8 Принципы построения параллельных вычислительных систем

8.1 Пути достижения параллелизма

В общем плане под параллельными вычислениями понимаются процессы обработки данных, в которых одновременно могут выполняться нескольких машинных операций. Достижение параллелизма возможно только при выполнимости следующих требований к архитектурным принципам построения вычислительной системы:

- *независимость функционирования отдельных устройств ЭВМ* – данное требование относится в равной степени ко всем основным компонентам вычислительной системы - к устройствам ввода-вывода, к обрабатывающим процессорам и к устройствам памяти;
- *избыточность элементов вычислительной системы* – организация избыточности может осуществляться в следующих основных формах:
- *использование специализированных устройств* таких, например, как отдельных процессоров для целочисленной и вещественной арифметики, устройств многоуровневой памяти (регистры, кэш-память различных уровней);

- *дублирование устройств ЭВМ* путем использования, например, нескольких однотипных обрабатывающих процессоров или нескольких устройств оперативной памяти.

Дополнительной формой обеспечения параллелизма может служить *конвейерная реализация обрабатывающих устройств*, при которой выполнение операций в устройствах представляется в виде исполнения последовательности составляющих операцию подкоманд; как результат, при вычислениях на таких устройствах могут находиться на разных стадиях обработки одновременно несколько различных элементов данных.

При рассмотрении проблемы организации параллельных вычислений следует различать следующие возможные режимы выполнения независимых частей программы:

- *многозадачный режим (режим разделения времени)*, при котором для выполнения процессов используется единственный процессор; данный режим является псевдопараллельным, когда активным (исполняемым) может быть один единственный процесс, а все остальные процессы находятся в состоянии ожидания своей очереди на использование процессора; использование режима разделения времени может повысить эффективность организации вычислений (например, если один из процессов не может выполняться из-за ожидании вводимых данных, процессор может быть задействован для готового к исполнению процесса); кроме того, в данном режиме проявляются многие эффекты параллельных вычислений (необходимость взаимоисключения и синхронизации процессов и др.) и, как результат, этот режим может быть использован при начальной подготовке параллельных программ;

- *параллельное выполнение,* когда в один и тот же момент времени может выполняться несколько команд обработки данных; данный режим вычислений может быть обеспечен не только при наличии нескольких процессоров, но реализуем и при помощи конвейерных и векторных обрабатывающих устройств;

- *распределенные вычисления;* данный термин обычно используют для указания параллельной обработки данных, при которой используется несколько обрабатывающих устройств, достаточно удаленных друг от друга и в которых передача данных по линиям связи приводит к существенным временным задержкам; как результат, эффективная обработка данных при таком способе организации вычислений возможна только для параллельных алгоритмов с низкой интенсивностью потоков межпроцессорных передач данных; перечисленные условия является характерными, например, при организации вычислений в многомашинных вычислительных комплексах, образуемых объединением нескольких отдельных ЭВМ с помощью каналов связи локальных или глобальных информационных сетей.

В рамках данного пособия основное внимание будет уделяться

второму типу организации параллелизма, реализуемому на многопроцессорных вычислительных системах.

8.2 Классификация вычислительных систем

Одним из наиболее распространенных способов классификации ЭВМ является систематика Флинна (Flynn), в рамках которой основное внимание при анализе архитектуры вычислительных систем уделяется способам взаимодействия последовательностей (потоков) выполняемых команд и обрабатываемых данных. В результате такого подхода различают следующие основные типы систем:

- *SISD (Single Instruction, Single Data)* – системы, в которых существует одиночный поток команд и одиночный поток данных; к данному типу систем можно отнести обычные последовательные ЭВМ;
- *SIMD (Single Instruction, Multiple Data)* – системы с одиночным потоком команд и множественным потоком данных; подобный класс систем составляют МВС, в которых в каждый момент времени может выполняться одна и та же команда для обработки нескольких информационных элементов;
- *MISD (Multiple Instruction, Single Data)* – системы, в которых существует множественный поток команд и одиночный поток данных; примеров конкретных ЭВМ, соответствующих данному типу вычислительных систем, не существует; введение подобного класса предпринимается для полноты системы классификации;
- *MIMD (Multiple Instruction, Multiple Data)* – системы с множественным потоком команд и множественным потоком данных; к подобному классу систем относится большинство параллельных многопроцессорных вычислительных систем.

Следует отметить, что, хотя систематика Флинна широко используется при конкретизации типов компьютерных систем, такая классификация приводит к тому, что практически все виды параллельных систем (несмотря на их существенную разнородность) относятся к одной группе MIMD. Как результат, многими исследователями предпринимались неоднократные попытки детализации систематики Флинна. Так, например, для класса MIMD предложена практически общепризнанная структурная схема, в которой дальнейшее разделение типов многопроцессорных систем основывается на используемых способах организации оперативной памяти в этих системах (рисунок 1.18). Данный поход позволяет различать два важных типа многопроцессорных систем – **multiprocessors** *(мультипроцессоры или системы с общей разделяемой памятью)* и **multicomputers** *(мультикомпьютеры или системы с распределенной памятью)*.

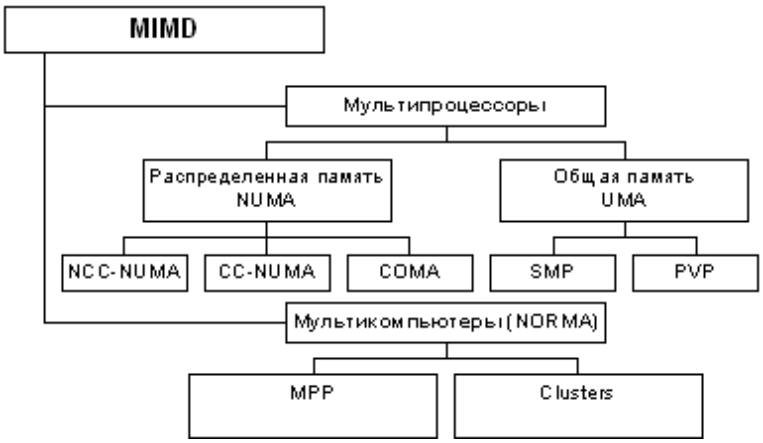


Рисунок 1.18 – Классификация систем класса MIMD

Далее для мультипроцессоров учитывается способ построения общей памяти. Возможный подход - использование единой (централизованной) общей памяти. Такой подход обеспечивает однородный доступ к памяти (uniform memory access or UMA) и служит основой для построения векторных суперкомпьютеров (parallel vector processor, PVP) и симметричных мультипроцессоров (symmetric multiprocessor or SMP). Среди примеров первой группы суперкомпьютер Cray T90, ко второй группе относятся IBM eServer p690, Sun Fire E15K, HP Superdome, SGI Origin 300 и др.

Общий доступ к данным может быть обеспечен и при физически распределенной памяти (при этом, естественно, длительность доступа уже не будет одинаковой для всех элементов памяти). Такой подход именуется как неоднородный доступ к памяти (non-uniform memory access or NUMA). Среди систем с таким типом памяти выделяют:

Системы, в которых для представления данных используется только локальная кэш-память имеющихся процессоров (cache-only memory architecture or COMA); примерами таких систем являются, например, KSR-1 и DDM;

Системы, в которых обеспечивается однозначность (когерентность) локальных кэш памяти разных процессоров (cache-coherent NUMA or CC-NUMA); среди систем данного типа SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000;

Системы, в которых обеспечивается общий доступ к локальной памяти разных процессоров без поддержки на аппаратном уровне когерентности кэша (non-cache coherent NUMA or NCC-NUMA); к данному типу относится, например, система Cray T3E.

Мультикомпьютеры (системы с распределенной памятью) уже не обеспечивают общий доступ ко всей имеющейся в системах памяти (no-remote memory access or NORMA). Данный подход используется при

построении двух важных типов многопроцессорных вычислительных систем - массивно-параллельных систем (massively parallel processor or MPP) и кластеров (clusters). Среди представителей первого типа систем - IBM RS/6000 SP2, Intel PARAGON/ASCI Red, транспьютерные системы Parsytec и др.; примерами кластеров являются, например, системы AC3 Velocity и NCSA/NT Super cluster.

Следует отметить чрезвычайно быстрое развитие *кластерного* типа многопроцессорных вычислительных систем. Под кластером обычно понимается множество отдельных компьютеров, объединенных в сеть, для которых при помощи специальных аппаратно-программных средств обеспечивается возможность унифицированного управления (single system image), надежного функционирования (availability) и эффективного использования (performance). Кластеры могут быть образованы на базе уже существующих у потребителей отдельных компьютеров либо же сконструированы из типовых компьютерных элементов, что обычно не требует значительных финансовых затрат. Применение кластеров может также в некоторой степени снизить проблемы, связанные с разработкой параллельных алгоритмов и программ, поскольку повышение вычислительной мощности отдельных процессоров позволяет строить кластеры из сравнительно небольшого количества (несколько десятков) отдельных компьютеров (lowly parallel processing). Это приводит к тому, что для параллельного выполнения в алгоритмах решения вычислительных задач достаточно выделять только крупные независимые части расчетов (coarse granularity), что, в свою очередь, снижает сложность построения параллельных методов вычислений и уменьшает потоки передаваемых данных между компьютерами кластера. Вместе с этим следует отметить, что организации взаимодействия вычислительных узлов кластера при помощи передачи сообщений обычно приводит к значительным временным задержкам, что накладывает дополнительные ограничения на тип разрабатываемых параллельных алгоритмов и программ.

8.3 Характеристика типовых схем коммуникации в многопроцессорных вычислительных системах

При организации параллельных вычислений в МВС для организации взаимодействия, синхронизации и взаимоисключения параллельно выполняемых процессов используется передача данных между процессорами вычислительной среды. Временные задержки при передаче данных по линиям связи могут оказаться существенными (по сравнению с быстродействием процессоров) и, как результат, коммуникационная трудоемкость алгоритма оказывает существенное влияние на выбор

параллельных способов решения задач.

Структура линий коммутации между процессорами вычислительной системы (топология сети передачи данных) определяется, как правило, с учетом возможностей эффективной технической реализации; немаловажную роль при выборе структуры сети играет и анализ интенсивности информационных потоков при параллельном решении наиболее распространенных вычислительных задач. К числу типовых топологий обычно относят следующие схемы коммуникации процессоров (рисунок 1.19):

- *полный граф* (completely-connected graph or clique)- система, в которой между любой парой процессоров существует прямая линия связи; как результат, данная топология обеспечивает минимальные затраты при передаче данных, однако является сложно реализуемой при большом количестве процессоров;

- *линейка* (linear array or farm) - система, в которой каждый процессор имеет линии связи только с двумя соседними (с предыдущим и последующим) процессорами; такая схема является, с одной стороны, просто реализуемой, а с другой стороны, соответствует структуре передачи данных при решении многих вычислительных задач (например, при организации конвейерных вычислений);

- *кольцо* (ring) - данная топология получается из линейки процессоров соединением первого и последнего процессоров линейки;

- *звезда* (star) - система, в которой все процессоры имеют линии связи с некоторым управляющим процессором; данная топология является эффективной, например, при организации централизованных схем параллельных вычислений;

- *решетка* (mesh) - система, в которой граф линий связи образует прямоугольную сетку (обычно двух- или трех- мерную); подобная топология может быть достаточно просто реализована и, кроме того, может быть эффективно используема при параллельном выполнении многих численных алгоритмов (например, при реализации методов анализа математических моделей, описываемых дифференциальными уравнениями в частных производных);

- *гиперкуб* (hypercube) - данная топология представляет частный случай структуры решетки, когда по каждой размерности сетки имеется только два процессора (т.е. гиперкуб содержит $2N$ процессоров при размерности N); данный вариант организации сети передачи данных достаточно широко распространен в практике и характеризуется следующим рядом отличительных признаков:

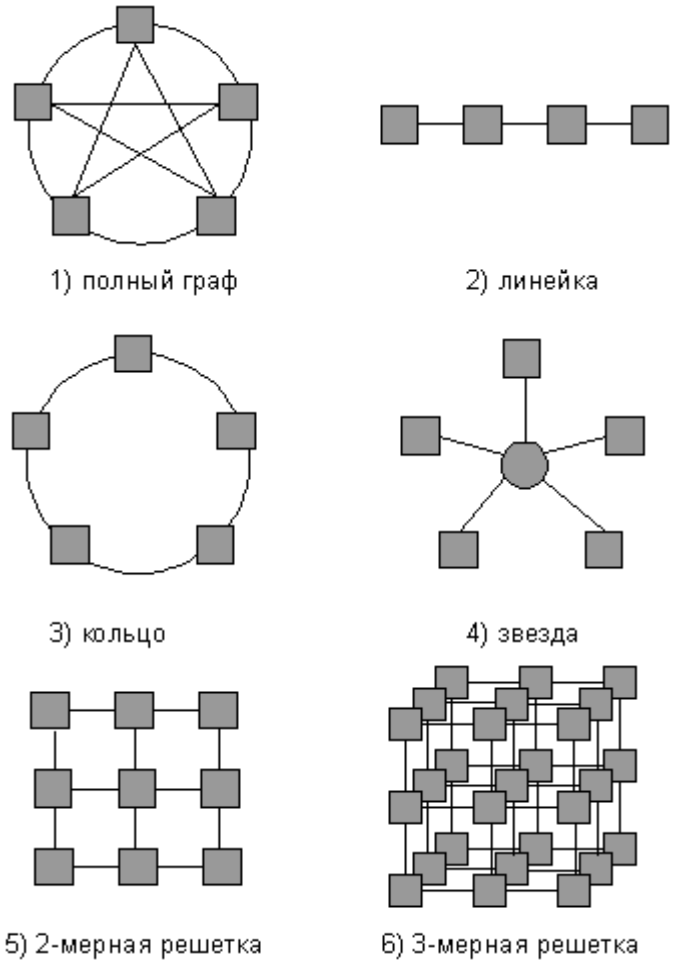


Рисунок 1.19 - Типовые топологии схем коммуникации процессоров

- два процессора имеют соединение, если двоичное представление их номеров имеет только одну различающуюся позицию;
- в *N*-мерном гиперкубе каждый процессор связан ровно с *N* соседями;
- *N*-мерный гиперкуб может быть разделен на два *(N–1)*-мерных гиперкуба (всего возможно *N* различных таких разбиений);
- кратчайший путь между двумя любыми процессорами имеет длину, совпадающую с количеством различающихся битовых значений в номерах процессоров (данная величина известна как расстояние Хэмминга).

II Организация компьютерных сетей

1 История развития компьютерных сетей

Концепция вычислительных сетей является логическим результатом эволюции компьютерной технологии. Первые компьютеры 50-х годов были большими, громоздкими и дорогими. Такие компьютеры не были предназначены для интерактивной работы, а использовались в режиме пакетной обработки.

Системы пакетной обработки, как правило, строились на базе мэйнфрейма – мощного и надежного компьютера универсального назначения. Подготавливались перфокарты, содержащие данные и команды программ, и передавались в вычислительный центр. Операторы вводили эти карты в компьютер, а распечатанные результаты пользователи получали обычно только на следующий день. Таким образом, одна неверно «набитая» карта означала как минимум суточную задержку.

Конечно, интерактивный режим работы, при котором можно с терминала оперативно руководить процессом обработки своих данных, был бы гораздо удобней. Но интересами программистов и пользователей на первых этапах развития вычислительных систем в значительной степени пренебрегали, поскольку пакетный режим – это самый эффективный режим использования вычислительной мощности, так как он позволяет выполнить в единицу времени больше задач, чем любые другие режимы. Наибольшее значение предавалось эффективности работы самого дорогого устройства вычислительной машины – процессора, в ущерб эффективности работы использующих его специалистов.

1.1 Многотерминальные системы – прообраз сети

По мере удешевления процессоров в начале 60-х годов появились новые способы организации вычислительного процесса, которые позволили учесть интересы пользователей. Начали развиваться интерактивные многотерминальные системы разделения времени (рисунок 2.1).

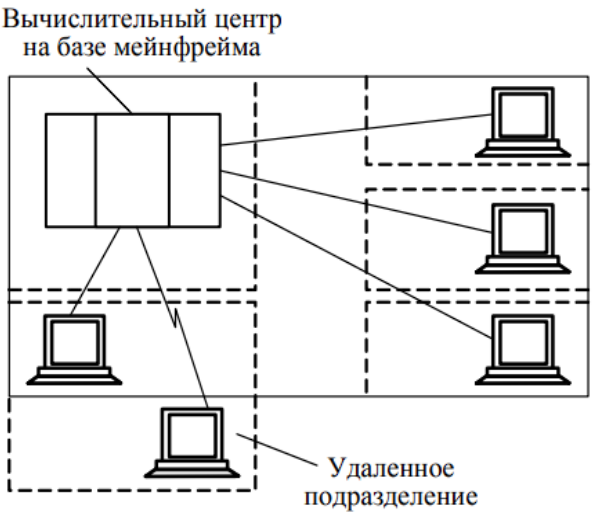


Рисунок 2.1 – Многотерминальная система – прообраз компьютерной сети

В таких системах компьютер использовали сразу несколько программистов-пользователей. Причем время реакции вычислительной системы было настолько малым, что пользователь не замечал параллельной работы с компьютером других программистов-пользователей.

Терминалы, выйдя за пределы вычислительного центра, рассредоточились по всему предприятию. И хотя вычислительная мощность оставалась полностью централизованной, некоторые функции – такие как ввод и вывод данных – стали распределенными. Такие многотерминальные централизованные системы внешне уже были очень похожи на локальные вычислительные сети. Действительно, рядовой программист-пользователь работу за терминалом мэйнфрейма воспринимал примерно так же, как сейчас он воспринимает работу за подключенным к сети персональным компьютером.

Таким образом, многотерминальные системы, работающие в режиме разделения времени, стали первым шагом на пути создания локальных вычислительных сетей. Но до появления локальных сетей нужно было пройти еще большой путь, так как многотерминальные системы, хотя и имели внешние черты распределенных систем, все еще сохраняли централизованный характер обработки данных. С другой стороны, и потребность предприятий в создании локальных сетей в это время еще не созрела – в одном здании просто нечего было объединять в сеть, так как из-за высокой стоимости вычислительной техники предприятия не могли себе позволить роскошь приобретения нескольких компьютеров. В этот

период был справедлив так называемый «закон Гроша», который эмпирически отражал уровень технологии того времени. В соответствии с этим законом производительность компьютера была пропорциональна квадрату его стоимости, отсюда следовало, что за одну и ту же сумму было выгоднее купить одну мощную машину, чем две менее мощных – их суммарная мощность оказывалась намного ниже мощности дорогой машины.

1.2 Появление глобальных сетей

Тем не менее, потребность в соединении компьютеров, находящихся на большом расстоянии друг от друга, к этому времени вполне назрела.

В 1962 году Американское агентство исследовательских проектов Министерства обороны США (Advanced Research Projects Agency of the U.S. Department of Defense, ARPA) начало реализацию проекта, который позднее получил название ARPANET и из которого позднее вырос современный Интернет (Internet).

В 1962 году важные исследования были начаты в ряде учебных заведений США и прежде всего в Массачусетском технологическом институте (MIT). Именно в 1962 году молодой американский ученый из MIT Дж. С. Ликлидер написал работу, где высказал идею глобальной сети, которая обеспечивала бы каждому жителю земли доступ к данным и программам из любой точки земного шара. В это же время другой ученый Л. Клейнрок закончил работу над своей докторской диссертацией в области теории коммуникационных сетей.

В 1963 году происходит важное событие: появляется первый универсальный стандарт ASCII – схема кодирования, назначающая численные значения-коды буквам, цифрам, знакам пунктуации и некоторым другим символам, в результате чего возникает возможность обмена информацией между компьютерами от различных изготовителей.

В 1964 году практически одновременно в MIT, RAND Corporation и Great Britain National Physical Laboratory (GBNPL) были развернуты работы по надежной передаче информации. Появилась идея коммутации пакетов, суть которой сводилась к тому, что любая информация, передаваемая по сети, разбивается на несколько частей (пакетов), которые затем независимо друг от друга перемещаются различными путями (маршрутами), пока не достигнут адресата. П. Бэран, Д. Дэвис, Л. Клейнрок параллельно вели исследования в этой области. П. Бэран был одним из первых, кто опубликовал свои исследования в статье «Передача данных в сетях».

В 1967 году произошло событие, которое сыграло важную роль в развитии сетевых технологий: модем, изобретенный в начале шестидесятых, был существенно усовершенствован Дж. Ван Гином из Станфордского научно-исследовательского института (Stanford Research Institute, SRI). Ученый предложил приемник, который мог надежно распознавать биты информации на фоне шумовых помех, создаваемых междугородними телефонными линиями.

В 1967 году Л. Робертс организовал научную конференцию в Анн-Арбор в штате Мичиган, на которую он пригласил основных разработчиков сетевого проекта. Конференция имела огромное значение – параллельно проводимые работы начали объединяться. Термин «ARPANET» впервые упоминался в ходе выступления Л. Робертса именно на этой конференции. На этой же конференции другой выдающийся ученый У. Кларк впервые высказал идею и предложил термин «IMP» – Interface Message Processors, обозначающий устройства для управления трафиком в сети, которые впоследствии эволюционировали в современные маршрутизаторы.

В 1968 году началась работа по созданию IMP и уже через один год, в 1969 году, заработала сеть ARPANET, охватившая все Западное побережье США.

В 1970 году наблюдался рост сети – каждый месяц добавляется новый узел. В этом же году произошло еще два важных события. Во-первых, Д. Ритчи и К. Томпсон из BelLabs закончили работу над созданием операционной системы UNIX. Во-вторых, в этом же году рабочая группа NWG (Network Working Group) под руководством С. Крокера завершила работу над протоколом NCP (Network Control Protocol), а еще годом позже закончила работу над протоколом эмуляции терминала (Telnet) и существенно продвинулась в работе над протоколом передачи файлов (FTP). В 1971 году BBN разработала новую платформу – так называемые TIP-устройства (Terminal IMP, Terminal Interface Processor), что обеспечило возможность входить на удаленные хосты, сделав, таким образом, ARPANET доступной большему числу пользователей.

В 1972 году сеть ARPANET была публично продемонстрирована. Однако в этом же году произошло еще, по крайней мере, два события, которые оказали огромное влияние на развитие компьютерных технологий: Р. Томильсон написал программу, позволяющую отправлять электронную почту по ARPANET, ввел обозначение «user@host» и использовал символ @, который позднее (с 1980 года) был закреплен в международном стандарте адресов электронной почты. Постепенно сеть ARPANET расширялась, и среди клиентов появились такие частные

организации, как BBN, Xerox PARC и MITRE Corporation, а также государственные – NASA's Ames Research Laboratories, National Bureau of Standards и Air Force Research Facilities.

В 1973 году фирма ARPA переименовывается в DARPA, где буква «D» указывает на Defense (защита, оборона). DARPA (Defense Advanced Research Projects Agency) – агенство перспективного планирования оборонных научно-исследовательских работ, центральная научно-исследовательская организация министерства обороны, основной целью которой является выдача рекомендаций по внедрению принципиально новых технологий для военной промышленности. Под руководством Б. Кана начинается весьма сложная работа по объединению сетей, имеющих разные интерфейсы, скорости передачи данных и размеры пакетов. По сути дела, это была работа по созданию межсетевого протокола. В сентябре 1973 году появилась первая публикация по новому протоколу TCP (Transmition Control Protocol). TCP/IP со временем стал одним из наиболее популярных протоколов сетевого взаимодействия и стандартом для реализации глобальных сетевых соединений в силу открытости, масштабируемости и за счет предоставления одинаковых возможностей глобальным и локальным сетям.

В 1977 году был анонсирован компьютер Apple II, и появление настольных компьютеров с потенциальной возможностью коммуникаций при помощи модемного подключения дало новый толчок развитию сетевых технологий и модемной индустрии. К началу 1978 года эксперимент ARPANET был практически. Годом позже появилась служба USENET4, которая стала одним из первых примеров клиент-серверной организации.

К концу семидесятых годов архитектура и протоколы TCP/IP приобрели современный вид. К этому времени агентство DARPA стало признанным лидером в разработке сетей с коммутацией пакетов. Дальнейшее развитие сетевых технологий, в том числе беспроводных радиосетей и спутниковых каналов связи, стимулировало активность DARPA в исследовании проблем межсетевого взаимодействия и реализации принципов Интернета в ARPANET. DARPA не делало тайны из своей деятельности в области развития технологий Интернета, поэтому различные научные группы проявляли интерес к разработкам технологии глобальной сети.

Свое начало Интернет берет от сети ARPANET, но чаще Интернет называют наследницей NSFNET – американской сети, объединившей ученых NSF (National Science Foundation), которая сотрудничала, объединялась с ARPANET, а затем поглотила ее.

Многие эксперты называют временем зарождения Интернета начало 80-х годов. В это время DARPA инициировало перевод машин, подсоединенных к его исследовательским сетям, на использование стека TCP/IP. В 1981 году IWG (Internet Working Group) в DARPA публикует документ, в котором говорится о полном переходе с протокола NCP (Network Control Protocol) на протокол TCP/IP. С этих пор ARPANET становится магистральной сетью Интернет и активно используется для многочисленных экспериментов с TCP/IP. Окончательный переход к технологии Интернет произошел в январе 1983 года: в этом году протокол TCP/IP был принят Министерством обороны США, а сеть ARPANET была разбита на две независимые части. Одна из них, предназначенная для научных целей, сохранила название ARPANET, а вторая, большая по масштабу сеть MILNET, отошла к военному ведомству.

Для того чтобы стимулировать использование новых протоколов в учебных заведениях, DARPA сделало реализацию TCP/IP широко доступной для университетских кругов. В это время многие исследователи использовали версию ОС Unix университета Беркли (штат Калифорния), называемую BSD Unix (от Berkeley Software Distribution). Благодаря тому, что DARPA в свое время субсидировала компанию BBN и университет в Беркли с целью реализации протоколов TCP/IP для использования вместе с популярной ОС Unix, более 90% компьютерных факультетов университетов адаптировали новую сетевую технологию, и версия BSD стала фактическим стандартом для реализаций стека протоколов TCP/IP. Было выпущено несколько версий BSD, каждая из которых добавляла в TCP/IP новые возможности, в том числе 4.2BSD (1983 год), 4.3BSD (1986 год), 4.3BSD Tahoe (1988 год), 4.3BSD Reno (1990 год), 4.4BSD (1993 год).

С 1985 года NSF реализовала программу создания сетей вокруг своих суперкомпьютерных центров. И в 1986 году создание опорной сети (56 Кбит/с) между суперкомпьютерными центрами NSF привело к появлению целого ряда региональных сетей, таких как JVNCNET, NYSERNET, SURANET, SDSCNET, BARRNET и других. Так появилась магистральная сеть NSFNET, которая в конце концов объединила все эти научные центры и связала их с ARPANET. Таким образом, NSFNET связала пять суперкомпьютерных центров и открыла доступ к мощным вычислительным ресурсам для широкого круга исследователей. Для уменьшения платы за использование междугородних линий связи решено было развивать систему региональных сетей, которая объединяет компьютеры внутри какого-то региона и имеет выходы на подобные сети поблизости. При такой конфигурации все компьютеры являются равноправными и имеют связь «по цепочке» через соседние компьютеры

как друг с другом, так и с суперкомпьютерами NSF. Таким образом, начиная с 1986 года можно говорить о становлении глобальной компьютерной сети Интернет.

В 1988 году Интернет становится международной сетью – к нему присоединяются Канада, Дания, Финляндия, Франция, Норвегия и Швеция. Годом позже сеть уже насчитывала 80 000 узлов; в ноябре к Интернету присоединились Австрия, Германия, Израиль, Италия, Япония, Мексика, Нидерланды, Новая Зеландия и Великобритания – и вскоре количество узлов в сети выросло до 160 000. В том же году появилась технология FDDI (Fiber Distributed Data Interface) – распределенный интерфейс передачи данных по волоконно-оптическим каналам.

Если Интернет – изобретение коллективное, то идею гипертекста и WWW связывают с именем конкретного человека. В 1989 году Т. Бернерс-Ли высказал идею гипертекста, которая и послужила толчком к созданию World Wide Web. Т. Бернерс-Ли написал программу Eniquire, которая стала прообразом будущей WWW. В том же 1989 году Т. Бернерс-Ли начал работу над глобальным проектом паутины, и всего два года спустя (в 1991 году) первые WWW-объекты были помещены в Интернет.

1.3 Первые локальные сети

В начале 70-х годов произошел технологический прорыв в области производства компьютерных компонентов – появились большие интегральные схемы. Их сравнительно невысокая стоимость и высокие функциональные возможности привели к созданию миникомпьютеров, которые стали реальными конкурентами мэйнфреймов. Даже небольшие подразделения предприятий получили возможность покупать для себя компьютеры. Мини-компьютеры выполняли задачи управления технологическим оборудованием, складом и другие задачи уровня подразделения предприятия. Таким образом, появилась концепция распределения компьютерных ресурсов по всему предприятию. Однако при этом все компьютеры одной организации по-прежнему продолжали работать автономно (рисунок 2.2).

Позднее предприятия и организации стали соединять свои миникомпьютеры вместе и разрабатывать программное обеспечение, необходимое для их взаимодействия. В результате появились первые локальные вычислительные сети (рисунок 2.3). Они еще во многом отличались от современных локальных сетей, в первую очередь, своими устройствами сопряжения.

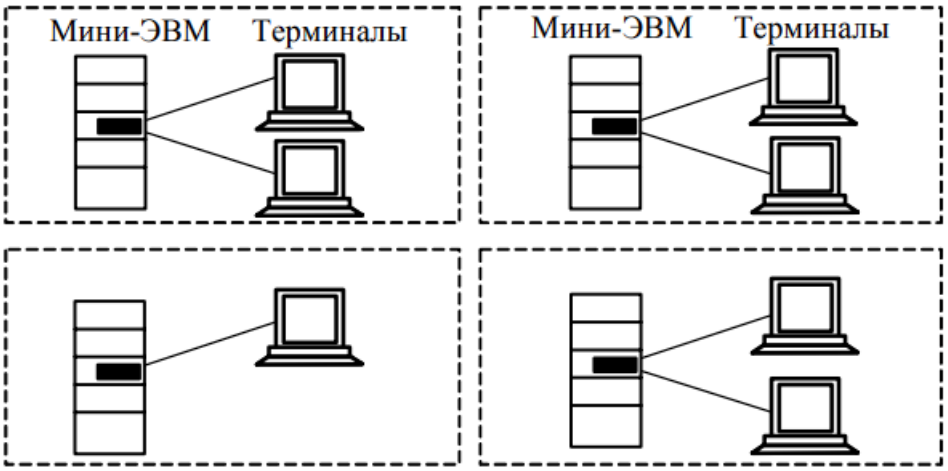


Рисунок 2.2 – Автономное использование мини-компьютеров на одном предприятии

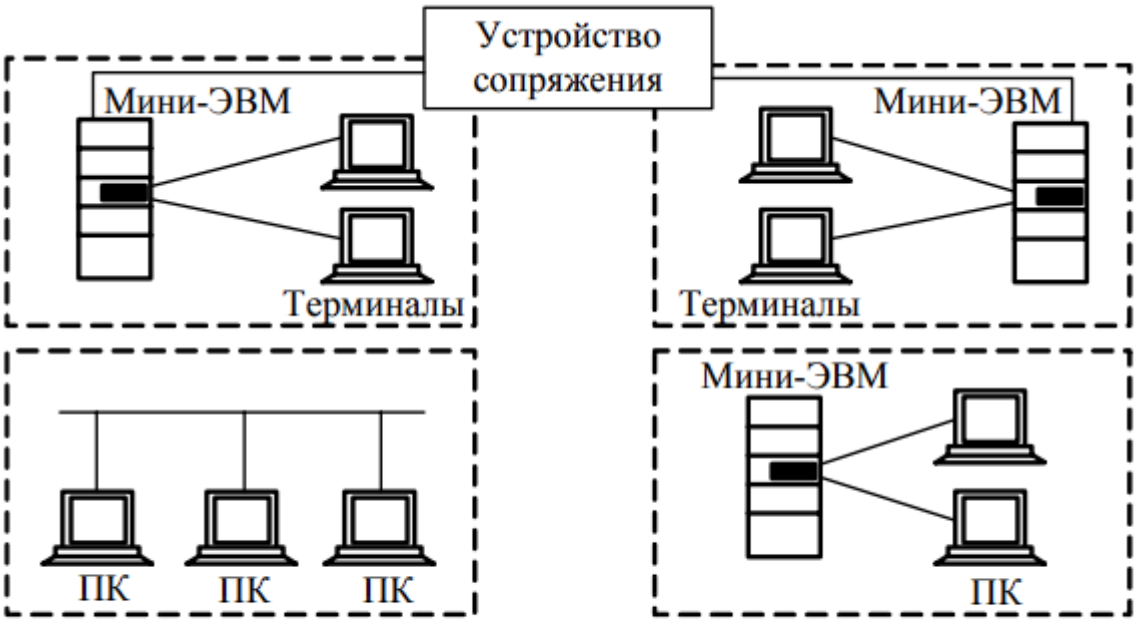


Рисунок 2.3 – Различные типы связей в первых локальных сетях

На первых порах для соединения компьютеров друг с другом использовались самые разнообразные нестандартные устройства со своим способом представления данных на линиях связи, своими типами кабелей и т. п.

1.4 Создание стандартных технологий локальных сетей

В середине 80-х годов положение дел в локальных сетях стало кардинально меняться. Утвердились стандартные технологии объединения компьютеров в сеть – Ethernet, Arcnet, Token Ring. Мощным стимулом для их развития послужили персональные компьютеры, которые стали преобладать в локальных сетях, причем не только в качестве клиентских компьютеров, но и в качестве центров хранения и обработки данных, то есть сетевых серверов, потеснив с этих привычных ролей мини-компьютеры и мэйнфреймы.

Стандартные сетевые технологии превратили процесс построения локальной сети из искусства в рутинную работу. Для создания сети достаточно было приобрести сетевые адаптеры соответствующего стандарта, например, Ethernet, стандартный кабель, адаптеры к кабелю стандартными разъемами и установить на компьютер одну из популярных сетевых операционных систем, например, NetWare.

Локальные сети в сравнении с глобальными внесли много нового в способы организации работы пользователей. Доступ к разделяемым ресурсам стал гораздо удобнее – пользователь мог просто просматривать списки имеющихся ресурсов, а не запоминать их идентификаторы или имена. После соединения с удаленным ресурсом можно было работать с ним с помощью уже знакомых пользователю по работе с локальными ресурсами команд. Возможность реализовать все нововведения разботчики локальных сетей получили в результате появления качественных кабельных линий связи, на которых даже сетевые адаптеры первого поколения обеспечивали скорость передачи данных до 10 Мбит/с.

Наибольшее распространение на тот момент получили телефонные каналы связи, но они были плохо приспособлены для высокоскоростной передачи дискретных данных – скорость в 1200 бит/с была для них хорошим достижением. Поэтому экономное расходование пропускной способности каналов связи часто являлось основным критерием эффективности методов передачи данных в глобальных сетях.

1.5 Современные тенденции развития компьютерных сетей

Сегодня компьютерные сети продолжают развиваться, причем достаточно быстро. Разрыв между локальными и глобальными сетями постоянно сокращается во многом из-за появления высокоскоростных территориальных каналов связи, не уступающих по качеству кабельным системам локальных сетей.

Изменяются и локальные сети. Вместо соединяющего компьютеры пассивного кабеля в них в большом количестве появилось разнообразное коммуникационное оборудование – коммутаторы, маршрутизаторы, шлюзы. Благодаря такому оборудованию появилась возможность построения больших корпоративных сетей, насчитывающих тысячи компьютеров и имеющих сложную структуру. Возродился интерес к крупным компьютерам, в основном из-за того, что после спада эйфории по поводу легкости работы с персональными компьютерами выяснилось, что системы, состоящие из сотен серверов, обслуживать сложнее, чем несколько больших компьютеров. Поэтому на новом витке эволюционной спирали мэйнфреймы стали возвращаться в корпоративные вычислительные системы, но уже как полноправные сетевые узлы, поддерживающие Ethernet или Token Ring, а также стек протоколов TCP/IP, ставший благодаря Internet сетевым стандартом де-факто.

Проявилась еще одна очень важная тенденция, затрагивающая в равной степени как локальные, так и глобальные сети. В них стала обрабатываться несвойственная ранее вычислительным сетям информация – голос, видеоизображения, рисунки. Это потребовало внесения изменений в работу протоколов, сетевых операционных систем и коммуникационного оборудования. Сложность передачи такой мультимедийной информации по сети связана с ее чувствительностью к задержкам при передаче пакетов данных, задержки обычно приводят к искажению такой информации в конечных узлах сети. Так как традиционные службы вычислительных сетей, такие как передача файлов или электронная почта, создают малочувствительный к задержкам трафик и все элементы сетей разрабатывались в расчете на него, то появление трафика реального времени привело к большим проблемам.

Сегодня эти проблемы решаются различными способами, в том числе и с помощью, специально рассчитанной на передачу различных типов трафика технологии ATM. Однако, несмотря на значительные усилия, предпринимаемые в этом направлении, до приемлемого решения проблемы пока далеко. И в этой области предстоит еще много сделать, чтобы достичь заветной цели – слияния технологий не только локальных и глобальных сетей, но и технологий любых информационных сетей – вычислительных, телефонных, телевизионных и т. п. Хотя эта идея многим кажется утопией, серьезные специалисты считают, что предпосылки для такого синтеза уже существуют, и их мнения расходятся только в оценке примерных сроков такого объединения – называются сроки от 10 до 25 лет. Причем считается, что основой для объединения послужит технология коммутации пакетов, применяемая сегодня в

вычислительных сетях, а не технология коммутации каналов, используемая в телефонии, что, наверно, должно повысить интерес к сетям этого типа.

2 Основные понятия и определения

Сеть – это совокупность объектов, образуемых устройствами передачи и обработки данных. Международная организация по стандартизации определила компьютерную сеть как последовательную бит-ориентированную передачу информации между связанными друг с другом независимыми устройствами. В общем случае различают два понятия сети: коммуникационная сеть и информационная сеть (рисунок 2.4).

Коммуникационная сеть предназначена для передачи данных, также она выполняет задачи, связанные с преобразованием данных. Коммуникационные сети различаются по типу используемых физических средств соединения.

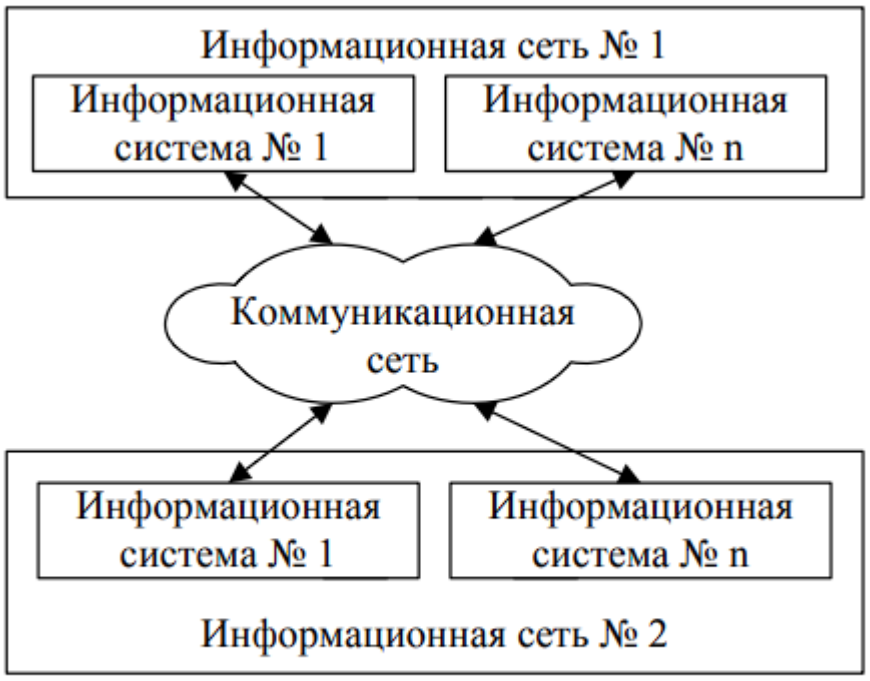


Рисунок 2.4 – Информационные и коммуникационные сети

Информационная сеть предназначена для хранения информации и состоит из информационных систем. На базе коммуникационной сети может быть построена группа информационных сетей.

Под *информационной системой* следует понимать систему, которая является поставщиком или потребителем информации.

Вычислительная сеть – это одна из разновидностей распределенных систем, предназначенная для распараллеливания вычислений, за счет чего может быть достигнуто повышение производительности и отказоустойчивости системы.

В общем, компьютерная сеть состоит из информационных систем и каналов связи.

Под *информационной системой* в данном случае следует понимать объект, способный осуществлять хранение, обработку или передачу информации. В состав информационной системы входят: компьютеры, программы, пользователи и другие составляющие, предназначенные для процесса обработки и передачи данных. В дальнейшем информационная система, предназначенная для решения задач пользователя, будет называться *рабочей станцией (client).* Рабочая станция в сети отличается от обычного персонального компьютера (ПК) наличием *сетевой карты (сетевого адаптера),* канала для передачи данных и сетевого программного обеспечения.

Под *каналом связи* следует понимать путь или средство, по которому передаются сигналы. Средство передачи сигналов называют *физическим каналом. Абонентский канал* – это физический канал, соединяющий коммуникационную сеть с абонентской системой. Параметры и характеристики абонентского канала в точке подключения системы определяются абонентским интерфейсом.

Каналы связи создаются по *линиям связи* при помощи сетевого оборудования и физических средств связи. Физические средства связи построены на основе витых пар, коаксиальных кабелей, оптических каналов или эфира. Между взаимодействующими информационными системами через физические каналы коммуникационной сети и узлы коммутации устанавливаются логические каналы.

Логический канал – это путь для передачи данных от одной системы к другой. Логический канал прокладывается по маршруту в одном или нескольких физических каналах. Логический канал можно охарактеризовать как маршрут, проложенный через физические каналы и узлы коммутации.

Информация в сети передается *блоками данных* по процедурам обмена между объектами. Эти процедуры называют *протоколами передачи данных.*

Протокол – это совокупность правил, устанавливающих формат и процедуры обмена информацией между двумя или несколькими устройствами.

Интерфейс – совокупность средств и методов взаимодействия между элементами или устройствами системы. Интерфейсы являются основой взаимодействия всех современных информационных систем. Если интерфейс какого-либо объекта (рабочей станции, сетевой карты, программы и т. д.) не изменяется (стандартизирован), это дает возможность модифицировать сам объект, не перестраивая принципы его взаимодействия с другими объектами.

Загрузка сети характеризуется параметром, называемым трафиком.

Трафик – это поток сообщений в сети передачи данных. Под ним понимают количественное измерение в выбранных точках сети числа проходящих блоков данных и их длины, выраженное в битах в секунду.

Существенное влияние на характеристику сети оказывает метод доступа.

Метод доступа – это способ определения того, как сеть управляет доступом к каналу связи (кабелю), что существенно влияет на ее характеристики. В сети все рабочие станции физически соединены между собою каналами связи по определенной структуре, называемой топологией.

Топология – это описание физических соединений в сети, указывающее какие рабочие станции могут связываться между собой. Тип топологии определяет производительность, работоспособность и надежность эксплуатации рабочих станций, а также время обращения к файловому серверу. В зависимости от топологии сети используется тот или иной метод доступа.

Состав основных элементов в сети зависит от ее архитектуры.

Архитектура – это концепция, определяющая взаимосвязь, структуру и функции взаимодействия рабочих станций в сети. Она предусматривает логическую, функциональную и физическую организацию технических и программных средств сети. Архитектура определяет принципы построения и функционирования аппаратного и программного обеспечения элементов сети.

3 Классификации компьютерных сетей

Локальные вычислительные сети (LAN, Local Area Network) – сети, которые имеют небольшие, локальные размеры, соединяют близкорасположенные компьютеры. Однако некоторые локальные сети легко обеспечивают связь на расстоянии нескольких десятков километров. Это уже размеры не комнаты, не здания, не близкорасположенных зданий, а может быть, даже целого города.

С другой стороны, по *глобальной сети (WAN, Wide Area Network или GAN, Global Area Network)* вполне могут связываться компьютеры, находящиеся на соседних столах в одной комнате.

Как правило, локальная сеть связывает от двух до нескольких десятков компьютеров. Но предельные возможности современных локальных сетей гораздо выше: максимальное число абонентов может достигать тысячи. Называть такую сеть малой не корректно.

В пределах одной сети могут использоваться как проводные кабели различных типов (витая пара, коаксиальный кабель, оптоволоконные кабели), так и радиоканалы.

Скорость передачи по локальной сети обязательно должна расти по мере роста быстродействия наиболее распространенных компьютеров.

Таким образом, главное отличие локальной сети от любой другой – высокая скорость передачи информации по сети. Но это еще не все, не менее важны и

другие факторы. В частности, принципиально необходим низкий уровень ошибок передачи, вызванных как внутренними, так и внешними факторами.

Особое значение имеет и такая характеристика сети, как возможность работы с большими нагрузками, то есть с высокой интенсивностью обмена (или, как еще говорят, с большим трафиком). Ведь если механизм управления обменом, используемый в сети, не слишком эффективен, то компьютеры могут подолгу ждать своей очереди на передачу.

Механизм управления обменом может гарантированно успешно работать только в том случае, когда заранее известно, сколько компьютеров (или абонентов, узлов) допустимо подключить к сети. Иначе всегда можно включить столько абонентов, что вследствие перегрузки «забуксует» любой механизм управления. Наконец, сетью можно назвать только такую систему передачи данных, которая позволяет объединять до нескольких десятков компьютеров, но никак не два, как в случае связи через стандартные порты.

Таким образом, сформулировать отличительные признаки локальной сети можно следующим образом:

− высокая скорость передачи информации, большая пропускная способность сети;

− низкий уровень ошибок передачи; допустимая вероятность ошибок передачи данных не должна превышать порядок 10^{-8}–10^{-12};

− эффективный, быстродействующий механизм управления обменом по сети;

− заранее четко ограниченное количество компьютеров, подключаемых к сети.

При таком определении понятно, что глобальные сети отличаются от локальных прежде всего тем, что они рассчитаны на неограниченное число абонентов. Кроме того, они используют (или могут использовать) не слишком качественные каналы связи и сравнительно низкую скорость передачи. А механизм управления обменом в них не может быть гарантированно быстрым.

Нередко выделяют еще один класс компьютерных сетей – *городские, региональные* сети (MAN, Metropolitan Area Network), которые обычно по своим характеристикам ближе к глобальным сетям, хотя иногда все-таки имеют некоторые черты локальных сетей, например, высококачественные каналы связи и сравнительно высокие скорости передачи. В принципе городская сеть может быть локальной со всеми ее преимуществами.

Сейчас нельзя провести четкую границу между локальными и глобальными сетями. Большинство локальных сетей имеет выход в глобальную. Но характер передаваемой информации, принципы организации обмена, режимы доступа к ресурсам внутри локальной сети, как правило, сильно отличаются от тех, что приняты в глобальной сети.

По локальной сети может передаваться самая разная цифровая информация: данные, изображения, телефонные разговоры, электронные письма и т. д. Кстати, именно задача передачи изображений, особенно полноцветных динамических, предъявляет самые высокие требования к быстродействию сети. Чаще всего локальные сети используются для разделения (совместного использования) таких ресурсов, как дисковое пространство, принтеры и выход в глобальную сеть, но это всего лишь незначительная часть тех возможностей, которые предоставляют средства локальных сетей.

4 Основные программные и аппаратные компоненты сети

Вычислительная сеть – это сложный комплекс взаимосвязанных и согласованно функционирующих программных и аппаратных компонентов. Изучение сети в целом предполагает знание принципов работы ее отдельных элементов:

− компьютеров;
− коммуникационного оборудования;
− операционных систем;
− сетевых приложений.

Весь комплекс программно-аппаратных средств сети может быть описан многослойной моделью. В основе любой сети лежит аппаратный слой стандартизованных компьютерных платформ.

В настоящее время в сетях широко и успешно применяются компьютеры различных классов – от персональных до мэйнфреймов и суперЭВМ. Набор компьютеров в сети должен соответствовать набору разнообразных задач, решаемых сетью.

Второй слой – это коммуникационное оборудование. Хотя компьютеры и являются центральными элементами обработки данных в сетях, в последнее время не менее важную роль стали играть коммуникационные устройства. Кабельные системы, повторители, мосты, коммутаторы, маршрутизаторы и модульные концентраторы из вспомогательных компонентов сети превратились в основные наряду с компьютерами и системным программным обеспечением как по влиянию на характеристики сети, так и по стоимости. Сегодня коммуникационное устройство может представлять собой сложный специализированный мультипроцессор, который нужно конфигурировать, оптимизировать и администрировать. Изучение принципов работы коммуникационного оборудования требует знакомства с большим количеством протоколов, используемых как в локальных, так и глобальных сетях.

Третьим слоем, образующим программную платформу сети, являются операционные системы (ОС). От того, какие концепции управления локальными и распределенными ресурсами положены в основу сетевой ОС, зависит

эффективность работы всей сети. При проектировании сети важно учитывать, насколько просто данная операционная система может взаимодействовать с другими ОС сети, насколько она обеспечивает безопасность и защищенность данных, до какой степени она позволяет наращивать число пользователей, можно ли перенести ее на компьютер другого типа и многие другие соображения.

Самым верхним слоем сетевых средств являются различные сетевые приложения, такие как сетевые базы данных, почтовые системы, средства архивирования данных, системы автоматизации коллективной работы и др. Очень важно представлять диапазон возможностей приложений для различных областей применения, а также знать, насколько они совместимы с другими сетевыми приложениями и операционными системами.

5 Модель OSI (Эталонная модель взаимодействия открытых систем)

Стандарты. Модель OSI, которая была определена в серии стандартов ISO/IEC 7498, состоит из следующих частей:

ISO/IEC 7498-1 — базовая модель;
ISO/IEC 7498-2 — архитектура безопасности;
ISO/IEC 7498-3 — наименования и адресация;
ISO/IEC 7498-4 — система менеджмента.

5.1 Основные принципы

Протоколы связи позволяют структуре на одном хосте взаимодействовать с соответствующей структурой того же уровня на другом хосте.

На каждом уровне N два объекта обмениваются блоками данных (PDU) с помощью протокола данного уровня на соответствующих устройствах. Каждый PDU содержит блок служебных данных (SDU), связанный с верхним или нижним протоколом.

Обработка данных двумя взаимодействующими OSI-совместимыми устройствами происходит следующим образом (рисунок 2.5)

1. Передаваемые данные составляются на самом верхнем уровне передающего устройства (уровень N) в протокольный блок данных (PDU).

2. PDU передается на уровень N-1, где он становится сервисным блоком данных (SDU).

3. На уровне N-1 SDU объединяется с верхним, нижним или обоими уровнями, создавая слой N-1 PDU. Затем он передается в слой N-2.

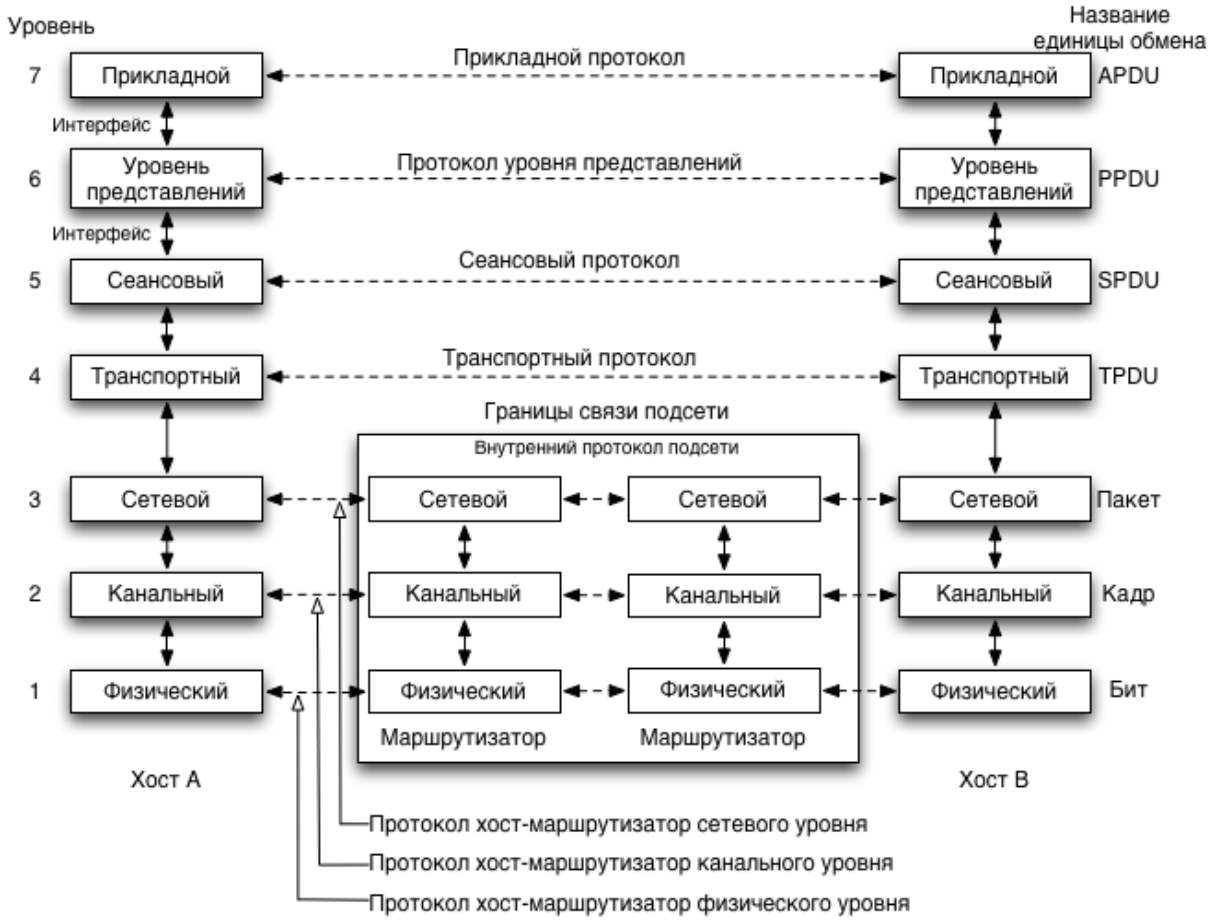


Рисунок 2.5 – Модель OSI (Эталонная модель взаимодействия открытых систем)

4. Процесс продолжается до достижения самого нижнего уровня, с которого данные передаются на принимающее устройство.

5. На приемном устройстве данные передаются от самого низкого уровня к самому высокому в виде серии SDU, последовательно удаляясь из верхнего или нижнего колонтитула каждого слоя до достижения самого верхнего уровня, где принимаются последние данные.

Наиболее часто принято начинать описание уровней модели OSI с седьмого уровня, называемого *прикладным*, на котором пользовательские приложения обращаются к сети. Модель OSI заканчивается 1-м уровнем — физическим, на котором определены стандарты, предъявляемые независимыми производителями к средам передачи данных:

- тип передающей среды (медный кабель, оптоволокно, радиоэфир и др.),
- тип модуляции сигнала,
- сигнальные уровни логических дискретных состояний (нули и единицы).

Любой протокол модели OSI должен взаимодействовать либо с протоколами своего уровня, либо с протоколами на единицу выше и/или ниже своего уровня. Взаимодействия с протоколами своего уровня называются горизонтальными, а с уровнями на единицу выше или ниже — вертикальными. Любой протокол модели OSI может выполнять только функции своего уровня и не может выполнять функций другого уровня, что не выполняется в протоколах альтернативных моделей.

Каждому уровню с некоторой долей условности соответствует свой операнд — логически неделимый элемент данных, которым на отдельном уровне можно оперировать в рамках модели и используемых протоколов: на физическом уровне мельчайшая единица — бит, на канальном уровне информация объединена в кадры, на сетевом — в пакеты (датаграммы), на транспортном — в сегменты. Любой фрагмент данных, логически объединённых для передачи — кадр, пакет, датаграмма — считается сообщением. Именно сообщения в общем виде являются операндами сеансового, представления и прикладного уровней.

К базовым сетевым технологиям относятся физический и канальный уровни.

5.2 Протоколы верхних уровней

Прикладной уровень. Прикладной уровень (уровень приложений; application layer) — верхний уровень модели, обеспечивающий взаимодействие пользовательских приложений с сетью; позволяет приложениям использовать сетевые службы:

- удалённый доступ к файлам и базам данных,
- пересылка электронной почты;
- отвечает за передачу служебной информации;
- предоставляет приложениям информацию об ошибках;
- формирует запросы к уровню представления.

Примеры протоколов прикладного уровня: RDP, HTTP, SMTP, SNMP, POP3, FTP, XMPP, OSCAR, Modbus, SIP, TELNET.

Определения протокола прикладного уровня и уровня представления очень размыты, и принадлежность протокола к тому или иному уровню, например, протокола HTTPS, зависит от конечного сервиса который предоставляет приложение.

В том случае, если протокол, например, HTTPS, используется для просмотра некой простой интернет-страницы через браузер, его можно рассматривать как протокол прикладного уровня. В том же случае, если протокол HTTPS используется как низкоуровневый протокол для

передачи финансовой информации, например, по протоколу ISO 8583, то протокол HTTPS будет являться протоколом уровня представления, а протокол ISO 8583 — протоколом уровня приложения. То же касается иных протоколов прикладного уровня.

Уровень представления. Уровень представления (presentation layer) обеспечивает преобразование протоколов и кодирование/декодирование данных. Запросы приложений, полученные с прикладного уровня, на уровне представления преобразуются в формат для передачи по сети, а полученные из сети данные преобразуются в формат приложений. На этом уровне может осуществляться сжатие/распаковка или шифрование/дешифрование, а также перенаправление запросов другому сетевому ресурсу, если они не могут быть обработаны локально.

Уровень представлений обычно представляет собой промежуточный протокол для преобразования информации из соседних уровней. Это позволяет осуществлять обмен между приложениями на разнородных компьютерных системах прозрачным для приложений образом. Уровень представлений обеспечивает форматирование и преобразование кода. Форматирование кода используется для того, чтобы гарантировать приложению поступление информации для обработки, которая имела бы для него смысл. При необходимости этот уровень может выполнять перевод из одного формата данных в другой.

Уровень представлений имеет дело не только с форматами и представлением данных, он также занимается структурами данных, которые используются программами. Таким образом, уровень 6 обеспечивает организацию данных при их пересылке.

Чтобы понять, как это работает, представим, что имеются две системы. Одна использует для представления данных расширенный двоичный код обмена информацией EBCDIC, например, это может быть мейнфрейм компании IBM, а другая — американский стандартный код обмена информацией ASCII (его использует большинство других производителей компьютеров). Если этим двум системам необходимо обменяться информацией, то нужен уровень представлений, который выполнит преобразование и осуществит перевод между двумя различными форматами.

Другой функцией, выполняемой на уровне представлений, является шифрование данных, которое применяется в тех случаях, когда необходимо защитить передаваемую информацию от доступа несанкционированными получателями. Чтобы решить эту задачу, процессы и коды, находящиеся на уровне представлений, должны

выполнить преобразование данных. На этом уровне существуют и другие подпрограммы, которые сжимают тексты и преобразовывают графические изображения в битовые потоки, так, что они могут передаваться по сети.

Стандарты уровня представлений также определяют способы представления графических изображений. Для этих целей может использоваться формат PICT — формат изображений, применяемый для передачи графики QuickDraw между программами.

Протоколы уровня представления: AFP — Apple Filing Protocol, ICA — Independent Computing Architecture, LPP — Lightweight Presentation Protocol, NCP — NetWare Core Protocol, NDR — Network Data Representation, XDR — eXternal Data Representation, X.25 PAD — Packet Assembler/Disassembler Protocol.

Сеансовый уровень. Сеансовый уровень (session layer) модели обеспечивает поддержание сеанса связи, позволяя приложениям взаимодействовать между собой длительное время. Уровень управляет созданием/завершением сеанса, обменом информацией, синхронизацией задач, определением права на передачу данных и поддержанием сеанса в периоды неактивности приложений.

Примеры протоколов сеансового уровня: H.245 (Call Control Protocol for Multimedia Communication), ISO-SP (OSI Session Layer Protocol (X.225, ISO 8327)), iSNS (Internet Storage Name Service), L2F (Layer 2 Forwarding Protocol), L2TP (Layer 2 Tunneling Protocol), NetBIOS (Network Basic Input Output System), PAP (Password Authentication Protocol), PPTP (Point-to-Point Tunneling Protocol), RPC (Remote Procedure Call Protocol), RTCP (Real-time Transport Control Protocol), SMPP (Short Message Peer-to-Peer), SCP (Session Control Protocol), ZIP (Zone Information Protocol), SDP (Sockets Direct Protocol).

5.3 Транспортная служба модели OSI

Транспортный уровень. Транспортный уровень (transport layer) модели предназначен для обеспечения надёжной передачи данных от отправителя к получателю. При этом уровень надёжности может варьироваться в широких пределах. Существует множество классов протоколов транспортного уровня, начиная от протоколов, предоставляющих только основные транспортные функции (например, функции передачи данных без подтверждения приёма), и заканчивая протоколами, которые гарантируют доставку в пункт назначения нескольких пакетов данных в надлежащей последовательности,

мультиплексируют несколько потоков данных, обеспечивают механизм управления потоками данных и гарантируют достоверность принятых данных. Например, UDP ограничивается контролем целостности данных в рамках одной датаграммы и не исключает возможности потери пакета целиком или дублирования пакетов, нарушение порядка получения пакетов данных; TCP обеспечивает надёжную непрерывную передачу данных, исключающую потерю данных или нарушение порядка их поступления или дублирования, может перераспределять данные, разбивая большие порции данных на фрагменты и наоборот, склеивая фрагменты в один пакет.

Примеры протоколов транспортного уровня: ATP (AppleTalk Transaction Protocol), CUDP (Cyclic UDP), DCCP (Datagram Congestion Control Protocol), FCP (Fibre Channel Protocol), IL (IL Protocol), NBF (NetBIOS Frames protocol), NCP (NetWare Core Protocol), SCTP (Stream Control Transmission Protocol), SPX (Sequenced Packet Exchange), SST (Structured Stream Transport), TCP (Transmission Control Protocol), UDP (User Datagram Protocol).

Более детально транспортная служба и транспортные протоколы рассмотрены в п.10 данного раздела.

5.4 Стек протоколов нижних уровней модели OSI

Сетевой уровень. Сетевой уровень (англ. network layer) модели предназначен для определения пути передачи данных. Отвечает за трансляцию логических адресов и имён в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию, отслеживание неполадок и «заторов» в сети.

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).

Примеры протоколов 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).

Канальный уровень. Канальный уровень (англ. data link layer) предназначен для обеспечения взаимодействия сетей на физическом

уровне и контроля ошибок, которые могут возникнуть. Полученные с физического уровня данные, представленные в битах, он упаковывает в кадры, проверяет их на целостность и, если нужно, исправляет ошибки (либо формирует повторный запрос повреждённого кадра) и отправляет на сетевой уровень. Канальный уровень может взаимодействовать с одним или несколькими физическими уровнями, контролируя и управляя этим взаимодействием.

Спецификация IEEE 802 разделяет этот уровень на два подуровня: MAC (англ. media access control) регулирует доступ к разделяемой физической среде, LLC (англ. logical link control) обеспечивает обслуживание сетевого уровня.

На этом уровне работают коммутаторы, мосты и другие устройства. Эти устройства используют адресацию второго уровня (по номеру уровня в модели OSI).

Примеры протоколов канального уровня: ARCnet, ATM, Controller Area Network (CAN), Econet, IEEE 802.3 (Ethernet), Ethernet Automatic Protection Switching (EAPS), Fiber Distributed Data Interface (FDDI), Frame Relay, High-Level Data Link Control (HDLC), IEEE 802.2 (предоставляет функции LLC для подуровня IEEE 802 MAC), Link Access Procedures, D channel (LAPD), IEEE 802.11 wireless LAN, LocalTalk, Multiprotocol Label Switching (MPLS), Point-to-Point Protocol (PPP), Point-to-Point Protocol over Ethernet (PPPoE), Serial Line Internet Protocol (SLIP), StarLan, Token ring, Unidirectional Link Detection (UDLD), X.25, ARP.

При разработке стеков протоколов на этом уровне решаются задачи помехоустойчивого кодирования. К таким способам кодирования относится код Хемминга, блочное кодирование, код Рида — Соломона.

В программировании этот уровень представляет драйвер сетевой платы, в операционных системах имеется программный интерфейс взаимодействия канального и сетевого уровней между собой. Это не новый уровень, а просто реализация модели для конкретной ОС. Примеры таких интерфейсов: ODI, NDIS, UDI.

Физический уровень. Физический уровень (англ. physical layer) — нижний уровень модели, который определяет метод передачи данных, представленных в двоичном виде, от одного устройства (компьютера) к другому. Составлением таких методов занимаются разные организации, в том числе: Институт инженеров по электротехнике и электронике, Альянс электронной промышленности, Европейский институт телекоммуникационных стандартов и другие. Осуществляют передачу электрических или оптических сигналов в кабель или в радиоэфир и,

соответственно, их приём и преобразование в биты данных в соответствии с методами кодирования цифровых сигналов.

На этом уровне также работают концентраторы, повторители сигнала и медиаконвертеры.

Функции физического уровня реализуются на всех устройствах, подключенных к сети. Со стороны компьютера функции физического уровня выполняются сетевым адаптером или последовательным портом. К физическому уровню относятся физические, электрические и механические интерфейсы между двумя системами. Физический уровень определяет такие виды сред передачи данных как оптоволокно, витая пара, коаксиальный кабель, спутниковый канал передач данных и т. п. Стандартными типами сетевых интерфейсов, относящимися к физическому уровню, являются: V.35, RS-232, RS-485, RJ-11, RJ-45, разъёмы AUI и BNC.

При разработке стеков протоколов на этом уровне решаются задачи синхронизации и линейного кодирования. К таким способам кодирования относится код NRZ, код RZ, MLT-3, PAM5, Манчестер II.

Примеры протоколов физического уровня: IEEE 802.15 (Bluetooth), IRDA, EIA RS-232, EIA-422, EIA-423, RS-449, RS-485, DSL, ISDN, SONET/SDH, 802.11 Wi-Fi, Etherloop, GSM Um radio interface, ITU и ITU-T, TransferJet, ARINC 818, G.hn/G.9960, Modbus Plus.

Краткая информация об уровнях модели OSI, типах данных, выполняемых функциях и протоколах реализации приведены в таблице 2.1.

Таблица 2.1 – Уровни модели OSI

Уровень (layer)	Тип данных (PDU)	Функции	Примеры протоколов/ оборудования
7. Прикладной (application)	Данные	Доступ к сетевым службам	HTTP, FTP, POP3, SMTP, WebSocket
6. Представления (presentation)		Представление и шифрование данных	ASCII, EBCDIC, SSL, gzip
5. Сеансовый (session)		Управление сеансом связи	RPC, PAP, L2TP, gRPC
4. Транспортный (transport)	Сегменты (segment) /Датаграммы (datagram)	Прямая связь между конечными пунктами и надёжность	TCP, UDP, SCTP
3. Сетевой (network)	Пакеты (packet)	Определение маршрута и логическая адресация	IPv4, IPv6, IPsec, AppleTalk, ICMP
2. Канальный (data link)	Биты (bit)/ Кадры (frame)	Физическая адресация и надёжность физического канала	PPP, IEEE 802.22, Ethernet, DSL, ARP, сетевая карта.
1. Физический (physical)	Биты (bit)	Работа со средой передачи, сигналами и двоичными данными	USB, RJ («витая пара», коаксиальный, оптоволоконный), радиоканал

6 Методы коммутации информации

Важное значение в архитектуре открытых сетей играет *коммутация информации.* Ее задачей является прокладка в сетях трактов, необходимых для доставки последовательностей блоков данных абонентским системам-адресатам. Существует несколько методов коммутации информации (рис. 4.1).

Нередко, особенно в управлении производственными процессами, требуется, чтобы каждый передаваемый блок данных был доставлен адресату с временной задержкой, не превышающей определенного максимума. Наряду с этим в большинстве случаев это требование отсутствует и блоки данных могут в разумных пределах приходить со случайным запаздыванием. Естественно, что в этом случае оборудование и программное обеспечение коммутации информации значительно проще.

В современных сетях используются четыре способа коммутации информации: коммутация каналов, пакетов, а также смешанная и интегральная коммутация.

Коммутация каналов появилась ранее других в сетях, имеющих большое число каналов. Для этой цели в коммуникационной подсети устанавливается один либо нужное число коммутаторов, именуемых *узлами коммутации каналов.* Каждый из них соединяет нужные пары каналов передачи данных. Благодаря этому создаются *тракты*—последовательности каналов, через которые обеспечивается взаимодействие пар абонентских систем.

Пример коммуникационной подсети, в которой работают четыре коммутатора (1-4), показан на рисунок 2.6. Каждый коммутатор выполняет функции связанные с реализаций протокола уровня 1 и обеспечением физических процессов (ФП).

Уровень 1 в коммутаторе соединяет рассматриваемый коммутатор с одним либо группой подходящих к нему каналов. Физический процесс осуществляет необходимую коммутацию, т. е. передает блоки данных из одного канала в другой. Так, блоки данных которые нужно передать (рисунок 2.5 слева) между абонентскими системами А, Е, проходят через тракт: система А—коммутатор 1 — узел 1 — узел 3 — система Е. Кроме этого, тракты, соединяющие системы А и Е могут быть проложены через коммутаторы 1, 2, 3 или 1, 4, 3. 1 любом случае образуется последовательность каналов, соединяющих взаимодействующие системы.

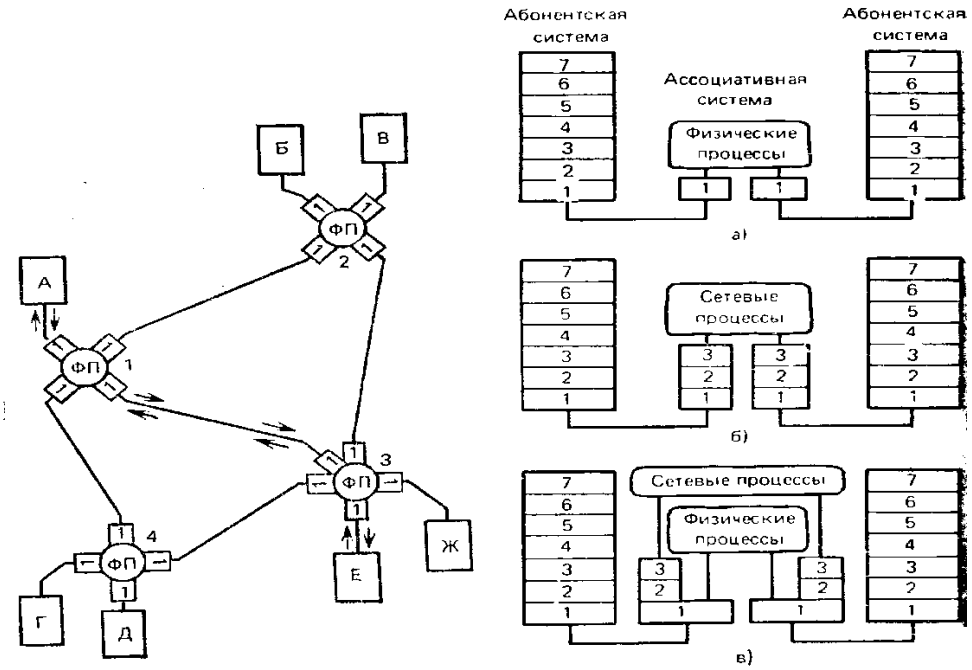


Рисунок 2.6 – Схемы коммутации каналов (а), коммутации пакетов (б) и смешанной коммутации (в)

Упрощенная схема коммутации каналов показана на рисунке 2.5(а), на котором, блоки данных проходят через коммутатор без обработки.

При работе в режиме коммутации каналов перед началом взаимодействия необходимо выполнить ряд предварительных процедур, связанных с организацией создания тракта, т. е. необходимой последовательности каналов. Для этого абонентская система—инициатор взаимодействия, должна сформировать и послать ближайшему коммутатору запрос на прокладку сквозь коммуникационную подсеть необходимого тракта. После образования тракта две абонентские системы (только две) используют его во время всего сеанса взаимодействия для передачи друг другу последовательностей блоков данных.

Как показывает опыт, во время сеанса последовательность каналов используется относительно небольшое время, остальное время каналы простаивают. Особенно во время диалога пользователя с прикладными процессами, когда он обдумывает полученные результаты и медленно (при помощи клавиатуры) передает очередной запрос.

Другим недостатком коммутации каналов является длительное время создания тракта, так как для этого необходимо дождаться момента, когда в коммуникационной подсети будет свободна вся необходимая последовательность каналов. Поэтому нередко при коротких сеансах время создания тракта превышает время передачи блоков данных. Все это приводит к тому, что коммутация каналов оказывается самым неэффективным способом коммутации информации. В этой связи он чаще всего используется при передаче больших массивов информации,

либо в тех случаях, когда требуется передача блоков данных с заданным временем их доставки, например, при передаче речи либо в технологических процессах.

Метод *коммутации пакетов* (рисунок 2.5(б)) свободен от указанных недостатков. Важной его особенностью является одновременное коллективное использование каналов значительным числом абонентских систем. Здесь физические тракты не создаются и ни один канал не отдается в монопольное владение парой абонентских систем. По каждому из каналов по мере поступления передаются блоки данных, посылаемые различными абонентскими системами.

Автономное функционирование каждого канала коммутационной подсети привели к тому, что каждый из них стал описываться двумя уровнями протоколов. Физический уровень (1), как и ранее, характеризует канал, а канальный уровень (2) обеспечивает управление передачей блоков данных по этому каналу. Для последовательной передачи блоков от одной абонентской системы к другой потребовалось введение в ассоциативных системах уровня 3.

В результате в подсети вместо одноуровневых коммутаторов стали использовать трехуровневые маршрутизаторы. Задачей сетевых процессов здесь является выполнение операций коммутации информации и прокладки маршрутов движения последовательностей блоков данных. Благодаря этим процессам по каждому каналу передаются в любой комбинации и очередности блоки данных, принадлежащие различным парам одновременно взаимодействующих абонентских систем. Загрузка каналов резко повысилась.

Однако коммутация пакетов не обеспечивает строго определенное время доставки последовательностей блоков данных. Поэтому использование этого способа в задачах, где это время нужно гарантировать, оказалось невозможным либо затрудненным.

Случайное время доставки блоков данных является недостатком метода коммутации пакетов. Однако его экономические преимущества заставляют искать пути расширения области его использования. Этому помогает постоянное увеличение скоростей обработки блоков данных в маршрутизаторах и создание физических средств соединения, не загружаемых, в среднем, более чем на 2/3 их теоретической пропускной способности.

Опыт и расчеты показывают, что экономически выгодно по одним и тем же каналам передавать любую информацию. Как ту, которая требует определенного времени доставки, так и ту, которая допускает появление случайных величин запаздывания. В этой связи возникла необходимость создания смешанной коммутации, обеспечивающей на базе одного и того же множества каналов коммутацию как каналов, так и пакетов.

При *смешанной коммутации* (рисунок 2.5(в)) используются уровни и процессы, применяемые как в коммутации каналов, так и в коммутации пакетов. При смешанной коммутации имеющиеся каналы отдаются в первую очередь для

создания трактов, соединяющих абонентские системы. Свободные при этом каналы не простаивают и используются для коммутации пакетов. Естественно, что в рассматриваемом случае в подсети устанавливаются комбинированные узлы, выполняющие роль как коммутаторов каналов, так и коммутаторов пакетов.

Следует отметить, что смешанная коммутация начинает широко использоваться для одновременной передачи по одним и тем же группам каналов данных и речи. Четвертым типом коммутации информации является *интегральная коммутация*. Она, как и смешанная, предназначена для обеспечения передачи информации с заданным и случайным временем доставки блоков данных. Однако интегральная коммутация отличается от смешанной тем, что здесь коммутация каналов и коммутация пакетов осуществляются *одновременно* в каждом физическом канале.

Для обеспечения интегральной коммутации в каждом таком канале прокладывается группа виртуальных каналов. Любой из них работает так, что у пары взаимодействующих абонентских систем создается впечатление, что они, используя виртуальный канал, передают блоки данных по отданному им физическому каналу.

Интегральная коммутация информации осуществляется различными способами. Один из них называется *асинхронным временным мультиплексированием* (АТМ). Сущность этого способа иллюстрируется рисунком 2.7, в соответствии с которым, для каждого физического канала сети время делится на повторяющиеся циклы (тайм-слоты). По каналу передаются разграничители, каждый из которых сообщает о начале очередного цикла. После этого в каждом цикле выделяются *n* интервалов времени, необходимых для создания *n* виртуальных каналов.

На базе временных интервалов создаются виртуальные каналы. Так, интервалы «Канал 1» в последовательности циклов образуют виртуальный канал 1, предоставляемый паре абонентских систем. Аналогично, из интервалов *i*, где *i=2,..., n,* образуется виртуальный канал *i.*

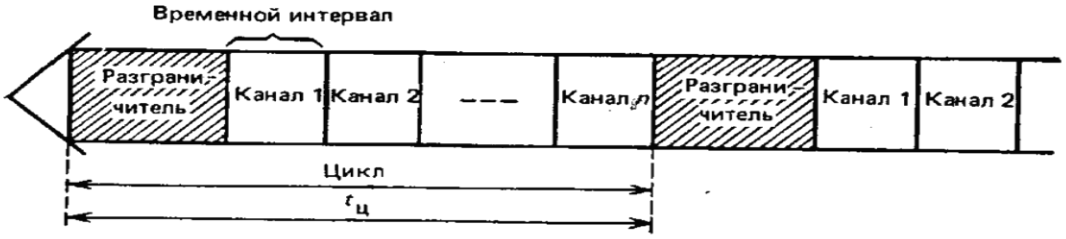
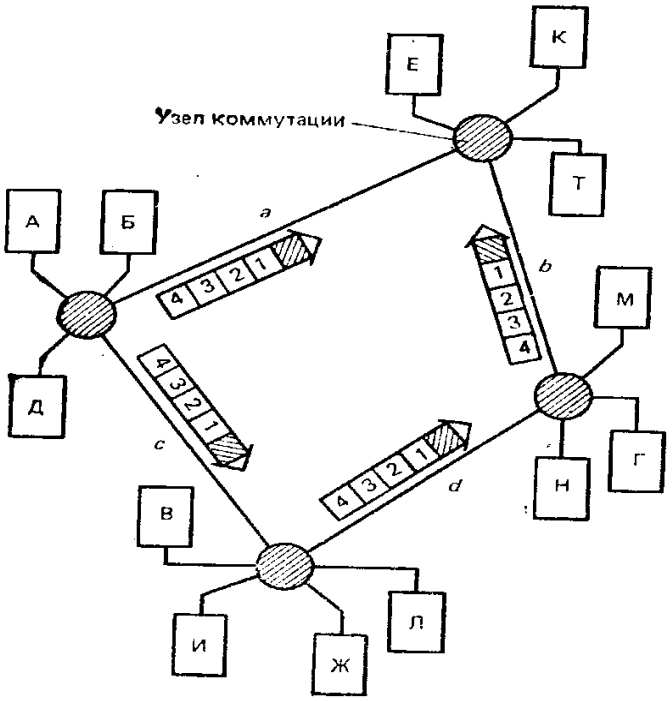


Рисунок 2.7 – Структура временного мультиплексирования

Рассмотрим пример, показанный на рисунке 2.8. В сети имеется четыре магистральных канала *a, b, c, d*, связывающие четыре узла (маршрутизатора). В этих каналах время разделено на циклы, каждый из которых содержит по четыре временных интервала.

Пусть необходимо организовать одновременное взаимодействие четырех пар абонентских систем *А-Е, Д-К, Д-М, И-Т.* Для этого: для систем *А-Е* в канале *a* на базе тайм-слотов 1 в каждом цикле организуется виртуальный канал *a1;* для систем *Д-К* в канале *a* на базе свободных тайм-слотов 2 в каждом цикле организуется виртуальный канал *a2;* для систем *Д-М* в каналах *c* и *d* на базе свободных тайм-слотов 1 в каждом цикле организуется виртуальный канал *c1+d1*; для систем *И-Т* в каналах *d* и *b* на базе свободных тайм-слотов 2 в канале *d* и свободных тайм-слотов 1 в канале *b* в каждом цикле организуется виртуальный канал *d2+b1*;



Номер канала	Взаимодей-ствующие абонентские системы	Виртуальные каналы	Номер канала	Взаимодействующие абонентские системы	Виртуальные каналы
1	А—Е	*a1*	3	Д--М	*c1+d1*
2	Д—К	*a2*	4	И—Т	*d2+b1*

Рисунок 2.8 – Структура временного мультиплексирования

Таким образом, системы *А-Е, Д-К, Д-М, И-Т.* получают через каждые $t_ц$ временной интервал для передачи блоков данных. В итоге происходит передача информации с детерминированным временем доставки.

После того как временные интервалы распределены, по запросам на коммутацию каналов осуществляется вторая часть управления коммутацией информации.

Оставшиеся временные интервалы используются для передачи в порядке очереди блоков данных, направляемых любыми абонентскими системами. Иначе говоря, в эти временные интервалы осуществляется коммутация пакетов. Так, в состоянии, показанном в таблице, для коммутации пакетов используются свободные участки: *a3*, *a4, b2, b3, b4, c2, c3*, *c4, d3*, *d4.* Естественно, что картина запросов на коммутацию каналов все время меняется. В соответствии с этим изменяется и список временных интервалов, оставляемых для коммутации пакетов.

Таким образом, при интегральной коммутации уже фактически нет в классическом понимании ни коммутации каналов, ни коммутации 'пакетов. Здесь оба вида коммутации слились в один способ передачи информации с гарантией либо без гарантии времени доставки блоков данных.

Важно отметить, что предыдущие способы коммутации информации оперировали со значительным (не менее трех) числом каналов в коммуникационной подсети. Что же касается интегральной коммутации, то она может осуществляться и при наличии в подсети только одного физического канала, например, моноканала. На базе интегральной коммутации предоставляется возможность передавать любые виды информации: данные, графику, факсимиле, речь и даже телевидение.

Между системами, расположенными в сети (рисунок 2.7), возможны и другие тракты взаимодействия. Так, системы *А, Е* могут взаимодействовать не только через физический канал *а,* но также через последовательность физических каналов *c, d, b.* Системы *Д, М* могут быть связаны через физические каналы *a, b.*

7 Способы организации виртуальных каналов и управления потоками данных

Виртуальный или логический канал (соединение) должен обеспечивать ряд базовых функций, которые в целом можно сформулировать следующим образом:

- доставку пакетов данных адресату;
- сохранение последовательности доставки пакетов;
- обнаружения и исправления ошибок доставки, используя при необходимости процедуру квитирования и повторную отсылку искаженных или пропавших пакетов;

- использование механизмов управления потоком для координации скорости передачи и приема.

Существуют различные способы вышеуказанных функций.

1. *Протокол с остановками и ожиданием.* При этой процедуре одновременно может передаваться только один пакет. После этого передающая сторона ждет подтверждения. Если поступит отрицательное подтверждение (когда таковое предусмотрено) или произойдет просрочка времени ожидания ответа, пакет передается повторно. Пакет сбрасывается из накопителя передающей стороны лишь после получения положительного подтверждения.

Этот протокол подходит для полудуплексной передачи, при которой передача сторон чередуется. Однако очевидно, что этот протокол снижает производительность в случае полного дуплекса (при независимой передаче в обоих направлениях), в частности, если время распространения сигнала по каналу значительно больше времени передачи пакета. Если время распространения пренебрежимо мало за счет небольшой протяженности канала либо за счет сравнительно небольшой скорости передачи, протокол с остановками и ожиданием не приводит к существенному снижению производительности.

2. *N-возвращения,* или *непрерывная передача.* Здесь кадры, если они имеются, передаются непрерывно без ожидания подтверждения. При получении отрицательного подтверждения или истечения установленного времени ожидания неподтвержденный пакет и все последующие кадры передаются вновь. Простой пример такой передачи представлен на рисунке 2.9, где показана передача пакетов из А в В с подтверждением ПТВ каждого кадра при его получении стороной В.

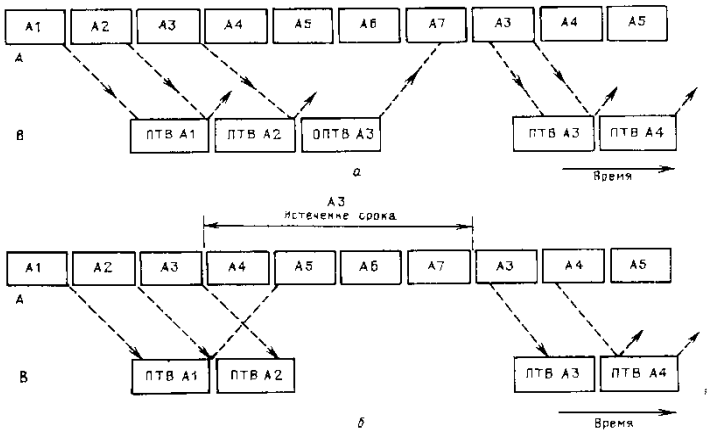


Рисунок 2.9 – Протокол с N-возвращениями: обработка подтверждения и сигнала истечения времени. Данные передаются только из А в В (стрелки указывают направления): (а) отрицательное подтверждение пакета 3, (б) истечение времени ожидания подтверждения

На рисунке 2.9, *а* предполагается, что пакет 3 принят с ошибкой, на что последовало отрицательное подтверждение ОПТЕ. На рисунке 2.9, *б* показан случай истечения времени ожидания подтверждения на стороне А. Для простоты показано только время ожидания подтверждения для пакета 3. Заметим, что *каждый* переданный пакет имеет такое же время ожидания подтверждения, и отсчет этого времени начинается сразу же после завершения передачи пакета; на рисунке 2.9 пакет 3 оказался единственным, для которого истекло время ожидания подтверждения. (Если в протоколе не используется отрицательное подтверждение, могут возникнуть две возможности: 1) в пункте В обнаружена ошибка пакета 3; пакет в пункте В сбрасывается, но больше ничего не предпринимается; 2) положительное подтверждение из пункта В исказилось при передаче; в этом случае пункт А повторяет передачу 3-го и всех последующих пакетов.)

Непрерывная передача пакетов очевидно увеличивает пропускную способность. В практических версиях протокола с N-возвращениями не все пакеты требуют подтверждения. Положительное подтверждение может служить подтверждением правильной передачи не только данного пакета, но и всех ему предшествовавших.

3. *Процедура, выборочного повторения.* В этом случае повторная передача требуется только для пакета, о котором поступило отрицательное подтверждение или для которого истекло время ожидания подтверждения. Это очевидно увеличивает пропускную способность по сравнению со случаем N-возвращений. Однако на приемном конце требуется накопитель с перестроениями, так как в этом случае кадры могут повторно передаваться и приниматься не по порядку. В частности, большие накопители требуются для спутниковой связи, где время распространения сигналов велико, а также для длинных магистралей, где в пути между передатчиком и приемником находятся много пакетов. По этой причине и из-за увеличения стоимости реализации протокол выборочного повторения (или, как иногда говорят, *селективного отказа)* не нашел коммерческого применения.

Во всех трех рассмотренных случаях подразумевалось, что нумерация пакетов не представляет никаких проблем. На практике же поле порядков номеров ограничено. Если передан пакет с максимальным порядковым номером, и на него не поступает подтверждение (это называется «дефицитом порядковых номеров»), то передатчик должен приостановить посылку пакетов. Новый пакет может быть послан лишь после получения подтверждения. Это явление снижает пропускную способность и может, в частности, вызвать осложнение в случае спутникового или другого канала большой протяженности.

8 Методы маршрутизации информации

Пакеты поступают в сеть передачи данных (СПД), имея в своем заголовке адрес порта назначения. Узел связи СПД, в который поступил пакет, должен по адресу порта назначения определить *маршрут* передачи пакета—выходную линию связи, в которую нужно передать пакет. При передаче данных по виртуальному каналу маршрутизация выполняется единственный раз, когда устанавливается виртуальное соединение. При передаче данных в форме дейтаграмм маршрутизация выполняется для каждого отдельного пакета.

Выбор маршрутов в узлах связи СПД производится по *алгоритму маршрутизации*—правилу назначения выходной линии связи на основе данных, содержащихся в заголовке пакета, и данных, представляющих состояние узла связи и, возможно, СПД в целом. Эффективность алгоритма маршрутизации характеризуется следующими показателями: 1) временем доставки пакетов; 2) нагрузкой, создаваемой на сеть потоками пакетов, поступающими в сеть и распределяемыми по линиям и узлам связи; 3) затратами ресурсов в узлах связи, в первую очередь — затратами памяти и времени процессора коммутационной ЭВМ. Первые два показателя — основные при оценке эффективности. Алгоритмы маршрутизации имеют целью обеспечить непрерывное продвижение пакетов от источников к адресатам. При этом алгоритм стремится выбрать наиболее подходящее направление передачи пакета — с минимальным временем доставки или наиболее полным использованием пропускной способности СПД. Эффективность алгоритмов маршрутизации уменьшается в связи со следующими факторами: 1) передачей пакета в направлении, не приводящем к минимальному времени доставки; 2) передачей пакета в узел связи, находящийся под высокой нагрузкой; 3) созданием дополнительной нагрузки на сеть за счет передачи служебной информации, необходимой для выполнения алгоритма маршрутизации.

Задача маршрутизации решается в следующих условиях. Сеть передачи данных имеет произвольную ячеистую структуру. Кратчайший маршрут, обеспечивающий доставку пакета за минимальное время, зависит от двух основных факторов:

1) топологии СПД и пропускной способности линий связи;

2) нагрузки на линии связи, определяемой числом пакетов, стоящих в очереди на передачу в каждом узле связи.

Топология сети изменяется в результате отказов линий и узлов связи и отчасти при развитии СПД—введении новых линий и узлов связи. Пропускная способность линий связи зависит от уровня помех и параметров аппаратуры, обслуживающей линии. Нагрузка на линии связи—наиболее динамичный фактор, изменяющийся крайне быстро и в труднопрогнозируемом направлении.

Следовательно, чтобы решение задачи было оптимальным, необходимо каждому узлу представлять информацию о состоянии СПД в целом — всех узлов и линий связи. Информация о текущей топологии сети и пропускной способности линий может быть представлена узлам. Однако не существует способа точно предсказать состояние нагрузки в сети. Поэтому алгоритмы маршрутизации могут использовать данные о состоянии нагрузки, запаздывающие по отношению к текущему моменту времени из-за конечной скорости передачи данных и не соответствующие последующим моментам времени, в которые будет принято решение о направлении передачи пакетов. Таким образом, во всех случаях алгоритмы маршрутизации работают в условиях неопределенности текущего и будущего состояния СПД и пульсирующей нагрузки, создаваемой абонентами.

Классификация алгоритмов маршрутизации производится в зависимости от направленности передачи пакетов и способов представления данных о топологии и нагрузке сети.

Простая маршрутизация — способ маршрутизации, не изменяющийся при изменении топологии и состояния СПД. Простая маршрутизация обеспечивается разными алгоритмами, типичными из которых являются алгоритмы случайной и лавинной маршрутизации.

Случайная маршрутизация — передача пакета из узла в любом, случайным образом выбранном направлении, кроме направления, по которому пакет поступил в узел. Пакет, совершая «блуждания» по сети, с конечной вероятностью когда-либо достигает адресата. Очевидно, что случайная маршрутизация неэффективна ни по времени доставки пакетов, ни по использованию пропускной способности сети.

Лавинная маршрутизация — передача пакета из узла во всех направлениях, кроме того, по которому поступил пакет. При этом, если узел связан с *n* Другими узлами СПД, пакет передается в *n*—1 направлениях, т. е. размножается. Очевидно, что хотя бы одно направление обеспечит доставку пакета за минимальное время, т. е. лавинная маршрутизация гарантирует малое время доставки, однако это достигается за счет резкого ухудшения использования пропускной способности сети из-за загрузки ее большим числом пакетов.

Маршрутизация по предыдущему опыту — передача пакета в направлении, выбираемом на основе анализа потока, проходящего через узел. При этом пакеты, поступая в сеть, снабжаются адресами получателя и источника и счетчиком числа пройденных узлов. Пакет, который пришел в узел со значением счетчика 1, определяет соседний узел; пакет со значением счетчика 2 определяет узел, находящийся на расстоянии двух шагов, и т. д. Эти данные, позволяют установить топологию сети и на ее основе построить таблицу для выбора маршрута. Постоянно анализируя ' число пройденных узлов, можно изменять таблицу маршрутов, если появился пакет с числом пройденных узлов, меньшим ранее

зарегистрированного. Этот способ маршрутизации позволяет узлам приспосабливаться к изменению топологии сети, однако процесс адаптации протекает медленно и неэффективно. Метод изучения пути передачи пакетов используется для построения ряда модификаций алгоритмов простой маршрутизации.

Простая маршрутизация, не обеспечивая направленной передачи пакетов от источников к адресатам, имеет низкую эффективность. Основное ее достоинство — обеспечение устойчивой работы СПД при выходе из строя различных частей сети.

Фиксированная маршрутизация — способ выбора направления передачи по таблице маршрутизации, устанавливающей направление передачи для каждого узла назначения. Таблицы маршрутизации определяют кратчайшие пути от узлов к адресатам и вводятся в узлы связи, например, от управляющего центра сети. Для слабозагруженных сетей этот способ маршрутизации дает хорошие результаты, но его эффективность падает по мере увеличения нагрузки на сеть. При отказе линий связи необходимо менять таблицу маршрутизации. Для этого можно, например, размещать в каждом узле связи набор таблиц маршрутизации, подготовленных на случай отказа одной из линий связи. При возникновении отказа по узлам сети рассылается управляющий пакет, содержащий сведения об отказе, реагируя на который, узлы меняют таблицы маршрутизации путем выбора соответствующих таблиц из хранимого набора. Очевидно, что разработать способ фиксированной маршрутизации, обеспечивающей работоспособность сети при отказе многих линий, является чрезвычайно трудной задачей. К тому же фиксированная маршрутизация не позволяет адаптироваться к изменениям нагрузки, что в общем случае приводит к значительным задержкам пакетов в СПД. Фиксированная маршрутизация может строиться на основе единственного пути передачи пакетов между двумя абонентами. Такой способ называется *однопутевой маршрутизацией.* Его недостаток — неустойчивость к отказам и перегрузкам. Для повышения устойчивости в таблицах маршрутизации указывается несколько возможных путей передачи пакета и вводится правило выбора целесообразного пути. Такой способ называется *многопутевой маршрутизацией.*

Адаптивная маршрутизация — способ выбора направления передачи, учитывающий изменение состояния СПД. При адаптивной маршрутизации узлы СПД принимают решение о выборе маршрутов, реагируя на разного рода данные об изменении топологии и нагрузки. В идеальном случае каждый узел сети для принятия решения должен располагать полной информацией о текущем состоянии всех остальных узлов, о топологии сети и длине очередей к каждому направлению в каждом узле. Однако, как показали исследования, даже в этом идеальном случае задержки в СПД лишь немногим меньше, чем при фиксированной маршрутизации, таблицы которой определяют кратчайшие пути в сети и не

изменяются при колебаниях нагрузки. Дело в том, что оптимальные маршруты, формируемые на основе самой свежей информации о распределении нагрузки в сети, становятся неоптимальными в последующие моменты времени, когда пакеты еще не достигли адресатов. Когда, например, сильнозагруженные узлы получают информацию о том, что некоторая часть сети загружена слабо, они одновременно направляют пакеты в эту часть сети, создавая в сети, быть может, худшую ситуацию, чем предшествующая. Таким образом, алгоритмы адаптивной маршрутизации не обеспечивают оптимальности маршрутов. Однако выбор даже не оптимального, а близкого к нему маршрута приводит к значительному уменьшению времени доставки, особенно при пиковых нагрузках, а также к некоторому увеличению пропускной способности сети. Поэтому адаптивная маршрутизация получила широкое применение в вычислительных сетях и в первую очередь—в сетях с большим числом узлов связи (10 и более).

Алгоритмы адаптивной маршрутизации классифицируются по информации, используемой ими для принятия решений при назначении маршрутов.

Локальная адаптивная маршрутизация основана на использовании информации, имеющейся в отдельном узле СПД. Эта информация включает в себя: 1) таблицу маршрутизации, определяющую все направления передачи пакетов; 2) данные о текущем состоянии выходных каналов (работают или не работают); 3) длину очередей пакетов, ожидающих передачи по выходным каналам. Информация о состояниях других узлов сети не используется. Таблицы маршрутизации указывают кратчайшие маршруты, проходящие через минимальное число узлов и обеспечивающие передачу пакета в узел назначения за минимальное время.

Распределенная адаптивная маршрутизация основана на использовании информации, получаемой от соседних узлов сети. Этот способ маршрутизации может реализоваться, например,

следующим образом. Каждый узел сети формирует таблицы маршрутов ко всем узлам назначения, минимизирующие задержки в сети, причем для каждого маршрута указывается фактическое время передачи пакета в узел назначения. До начала работы сети это время оценивается исходя из топологии сети. В процессе работы сети узлы регулярно обмениваются с соседними узлами таблицами задержки. После обмена каждый узел пересчитывает задержки с учетом поступивших данных и длины очередей в самом узле. Полученные значения используются для выбора маршрутов: пакет ставится в очередь к маршруту, который характеризуется минимальным временем доставки. Обмен таблицами задержки производится периодически или когда обнаруживаются существенные изменения задержки из-за изменения очередей на передачу или состояния линий связи в результате отказа. Естественно, что периодический обмен таблицами задержки значительно увеличивает загрузку сети, а асинхронный — снижает.

Однако в каждом случае загрузка остается весьма существенной и к тому же сведения об изменении состояния узлов медленно распространяются по сети. Так, при обмене с интервалом 2/3 с время передачи данных составляет несколько секунд, и в этот период узлы направляют пакеты по старым путям, что может создать перегрузку в районе вышедших из строя компонентов сети.

Централизованная адаптивная маршрутизация основана на использовании информации, получаемой от центра маршрутизации. При этом каждый узел сети формирует сообщения о своем состоянии—длине очередей, работоспособности линий связи и эти сообщения передаются в центр маршрутизации. Последний на основе полученных данных формирует таблицы маршрутизации, рассылаемые всем узлам сети. Неизбежные временные задержки при передаче данных в центр маршрутизации, формировании и рассылке таблиц приводят к потере эффективности централизованной маршрутизации, особенно в ситуациях, когда нагрузка сильно пульсирует. Поэтому централизованная маршрутизация по эффективности не превосходит локальную адаптивную, а кроме того, отличается специфичным недостатком — потерей управления сетью при отказе центра маршрутизации.

Гибридная адаптивная маршрутизация основана на использовании таблиц, периодически рассылаемых центром маршрутизации, в сочетании с анализом длины очередей в узлах. Если таблица маршрутизации, сформированная для узла связи центром, определяет единственное направление передачи пакета, то пакет передается именно в этом направлении. Если же таблица определяет несколько направлений, то узел выбирает направление в зависимости от текущих значений длины очередей — по алгоритму локальной адаптивной маршрутизации. Гибридная маршрутизация компенсирует недостатки централизованной и локальной: маршруты, формируемые центром, являются устаревшими, но соответствуют глобальному состоянию сети; локальные алгоритмы являются «близорукими», но обеспечивают своевременность решений.

Протокол маршрутной информации (Routing Information Protocol) — один из самых простых протоколов динамической маршрутизации. Применяется в небольших компьютерных сетях, позволяет маршрутизаторам динамически обновлять маршрутную информацию (направление и дальность в хопах), получая ее от соседних маршрутизаторов.

Алгоритм маршрутизации RIP (алгоритм Беллмана — Форда) был впервые разработан в 1969 году, как основной для сети ARPANET.

RIP — так называемый протокол дистанционно-векторной маршрутизации, который оперирует транзитными участками (хоп, hop) в качестве метрики маршрутизации. Максимальное количество транзитных участков, разрешенное в RIP — 15 (метрика 16 означает «бесконечно большую метрику»). Каждый RIP-

маршрутизатор по умолчанию вещает в сеть свою полную таблицу маршрутизации раз в 30 секунд, довольно сильно нагружая низкоскоростные линии связи. RIP работает в сетях TCP/IP, используя UDP порт 520.

В современных сетевых средах RIP — не самое лучшее решение для выбора в качестве протокола маршрутизации, так как его возможности уступают более современным протоколам, таким как EIGRP, OSPF. Ограничение на 15 транзитных участков не дает применять его в больших сетях. Преимущество этого протокола — простота конфигурирования.

OSPF (Open Shortest Path First) — протокол динамической маршрутизации, основанный на технологии отслеживания состояния канала (link-state technology) и использующий для нахождения кратчайшего пути алгоритм Дейкстры.

Протокол OSPF был разработан IETF в 1988 году. Последняя версия протокола представлена в RFC 2328. Протокол OSPF представляет собой протокол внутреннего шлюза (Interior Gateway Protocol — IGP). Протокол OSPF распространяет информацию о доступных маршрутах между маршрутизаторами одной автономной системы.

OSPF имеет следующие преимущества:

- высокая скорость сходимости по сравнению с дистанционно-векторными протоколами маршрутизации;
- поддержка сетевых масок переменной длины (VLSM, см. п.9.2.3);
- оптимальное использование пропускной способности с построением дерева кратчайших путей.

Принцип работы заключается в следующем:

1. После включения маршрутизаторов протокол ищет непосредственно подключенных соседей и устанавливает с ними «дружеские» отношения.

2. Затем они обмениваются друг с другом информацией о подключенных и доступных им сетях. То есть они строят карту сети (топологию сети). Данная карта одинакова на всех маршрутизаторах.

3. На основе полученной информации запускается алгоритм SPF (Shortest Path First, «выбор наилучшего пути»), который рассчитывает оптимальный маршрут к каждой сети. Данный процесс похож на построение дерева, корнем которого является сам маршрутизатор, а ветвями — пути к доступным сетям. Данный процесс, то есть конвергенция, происходит очень быстро.

9 Принципы межсетевого взаимодействия

Межсетевым взаимодействием называется взаимодействие, осуществляемое системами, относящимися к разным сетям передачи данных. Межсетевое взаимодействие осуществляется в интересах взаимного использования

вычислительных, информационных и связных ресурсов разных сетей. В данном параграфе даны характеристики двух общепринятых подходов обеспечения межсетевого взаимодействия:

- объединение сетей передачи общего пользования (сетей X.25) в соответствии с Рекомендацией МККТТ X.75;

- объединение сетей коммутации пакетов в соответствии с принципами «объединенной сети» архитектуры протоколов DARPA.

Второй подход нашел также свое отражение в рекомендациях, регламентирующих службы сетевого уровня без установления соединения. Большинство практических работ по организации межсетевого взаимодействия в общем укладываются в одну из этих двух главенствующих концепций.

Поскольку конечной целью при взаимном объединении сетей является обеспечение связи функционирующих в разных ЭВМ пользовательских процессов, ключевым вопросом является согласование протоколов передачи данных, образующих систему связи между процессами. Если при подключении новой ГВМ (главной вычислительной машине, хосту) к уже действующей сети ЭВМ не возникает вопросов по организации связи (во вновь подключаемой ГВМ необходимо реализовать всю совокупность стандартных для данной сети протоколов), то при взаимном соединении двух (или более) сетей, когда также требуется, чтобы процессы ГВМ имели общую систему связи, этот вопрос становится критичным. С одной стороны, необходимо обеспечить достаточный уровень автономии уже действующих систем, т. е. минимально воздействовать на их характеристики, режимы работы, виды оказываемых ими услуг и т. д. С другой стороны, необходимо обеспечить межсетевое взаимодействие ГВМ. Этого можно достичь либо путем преобразования одного и более уровней под новые протоколы, либо путем преобразования протоколов между парой систем связи в их точках соприкосновения. Физически две или несколько сетей соединяются между собой с помощью устройства (или пары устройств), называемого межсетевым шлюзом. Для каждой из сопрягаемых сетей шлюз «выглядит», как обычная ГВМ этой сети (рисунок 2.10).

Различаются шлюзы двух типов. Шлюзы одного типа только считывают сообщения, поступающие из одной сети, определяют маршрут последующей передачи сообщений и передают сообщения в другую сеть, предварительно вложив их в упаковку уже этой сети. Так как сопрягаемые сети могут использовать различные средства связи (арендуемые линии связи, радиосвязь и т.д.), то шлюзы такого типа называются шлюзами с преобразованием носителя информации или трансляторами.

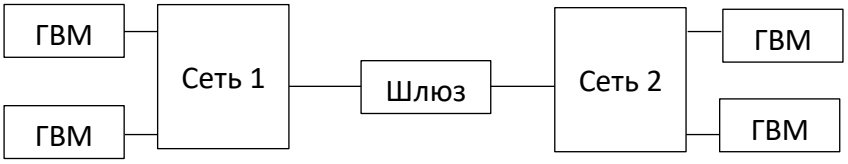


Рисунок 2.10 – Соединение сетей с помощью межсетевых шлюзов

Шлюзы другого типа осуществляют преобразование одного или нескольких протоколов, используемых в одной сети, в протокол (протоколы) другой сети путем замены сообщений, принятых из одной сети, другими сообщениями, в которых сохраняются те же самые семантические конструкции протокола. Такой тип шлюза называется межсетевым шлюзом с преобразованием протоколов.

Различие этих типов шлюзов состоит в различии уровней, на которых производится устранение расхождения в правилах передачи данных: для трансляторов — это уровень канала передачи данных и физический уровень, а для шлюзов с преобразованием протоколов—сетевой и (или) более высокие уровни. В первом случае принципиальных проблем, как правило, нет. При согласовании сетевых и транспортных протоколов проблемы несоответствия принципов адресования, размеров передаваемых сообщений, управления потоком и т. д. становятся критическими. Так, чтобы обеспечить сопряжение сетей с разной максимальной длиной пакетов, на шлюзах необходимо выполнять фрагментацию пакетов для их передачи через сеть с небольшим размером сообщений, а на ГВМ-получателе — восстанавливать исходное сообщение из фрагментов. Преобразование адресов также является сложной проблемой, если диапазон изменения адресов в одной сети не совпадает с диапазоном изменения в другой сети. Преобразование транспортных протоколов усложняется различиями в механизмах управления потоком: в одном протоколе единицей управления может быть октет, а в другом — письмо; в одних протоколах есть механизм предварительного выделения ресурсов, но нет управления интенсивностью передачи, а в других наоборот и т. д.

Функции согласования протоколов выполняет специальный межсетевой протокол, который является стандартом в объединенной сети. Шлюз рассматривается как состоящий из нескольких частей (по количеству сопрягаемых сетей), каждая из которых преобразует протокол конкретной сети в стандартный протокол и наоборот.

Стандартные межсетевые протоколы сетевого уровня, как правило, относятся к одному из двух типов: протоколы, регламентирующие взаимодействие с

установлением соединения, и протоколы взаимодействия без установления соединения.

9.1 Межсетевое взаимодействие посредством протокола X.75

Сети передачи общего пользования, соответствующие Рекомендации МККТТ X.25, должны соединяться посредством интерфейса, определенного в Рекомендации МККТТ X.75 (рисунок 2.11). Поскольку Рекомендация X.25 описывает работу виртуального соединения, сопряжение в соответствии с протоколом X.75 следует отнести к сопряжению протоколов с установлением соединения. Интерфейс между двумя сетями общего пользования, определенный в Рекомендации X.75, похож на интерфейс Рекомендации X.25. Оборудование на обеих сторонах этого интерфейса называется оконечным оборудованием сигнализации (Signaling Terminal Equipment — STE). Соединение STE/STE представляет собой шлюз, разделенный на части, каждая из которых является физическим устройством, управляемым той сетью, которая соединена с ней (рисунок 2.11).

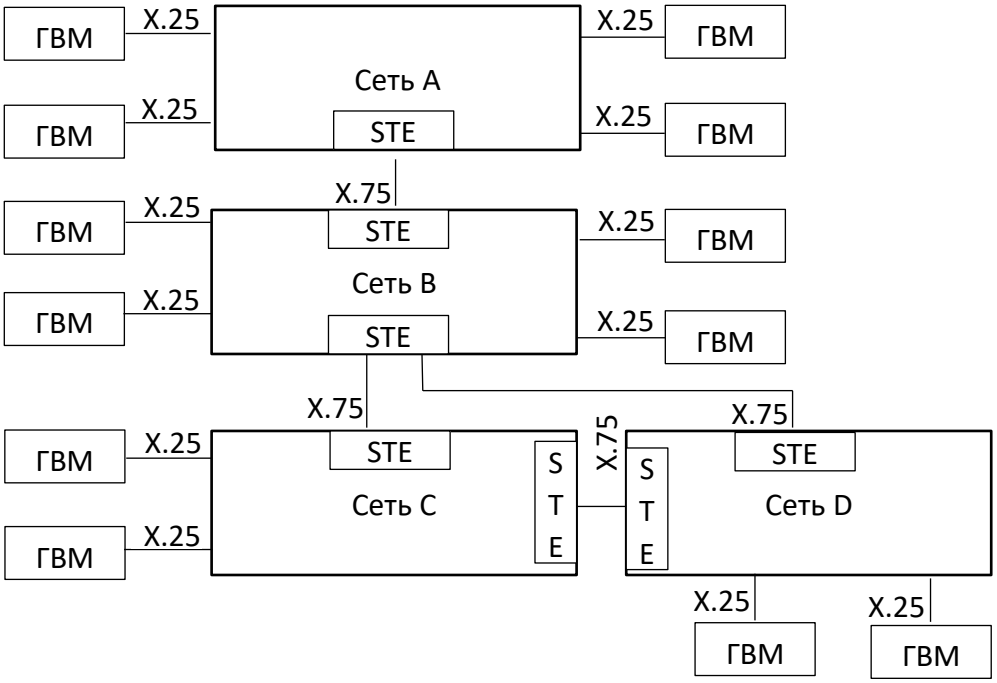


Рисунок 2.11 – Межсетевое взаимодействие посредством протокола X.75

Соединение сетей общего, пользования посредством интерфейса X.75 осуществляется через последовательность нескольких виртуальных соединений, каждое со своими процедурами управления потоком, защитой от ошибок и т. д. (рисунок 2.11).

9.2 Подход соединения сетей в рамках архитектуры протоколов DARPA

Подход соединения сетей в рамках архитектуры протоколов DARPA, состоит в следующем: различные сети (далее в данном параграфе объединяемые сети будут называться подсетями) соединены шлюзами, обеспечивающими преобразование протоколов на сетевом уровне. Стандартным протоколом сетевого уровня в объединенной сети служит межсетевой протокол IP). Помимо шлюзов IP должен быть реализован в каждой ГВМ, включенной в объединенную сеть. Шлюз представляет собой устройство, соединяющее несколько подсетей и выступающее для каждой подсети как ГВМ. В каждой подсети шлюз адресуется по тем же правилам, что и любая другая ГВМ этой подсети. На рисунке 2.12 изображена схема модели взаимного соединения подсетей.

Все ГВМ объединенной сети обмениваются стандартными пакетами — межсетевыми дейтаграммами. Адресная информация содержится в заголовке межсетевой дейтаграммы. ГВМ-отправитель подготавливает дейтаграмму, которая содержит заголовок и исходное сообщение, а затем выбирается шлюз собственной подсети (подсети, к которой подключена ГВМ-отправитель), который будет использоваться для дальнейшей передачи дейтаграммы через объединенную сеть.

Далее ГВМ-отправитель посылает дейтаграмму, вложенную в пакет подсети (данный пакет генерируется модулем протокола сетевого уровня данной подсети), к этому шлюзу. Шлюз принимает пакет подсети, извлекает из него дейтаграмму, анализирует заголовок IP, определяет адрес следующего шлюза (или адрес получателя) в одной из подсетей, с которыми он непосредственно соединен.

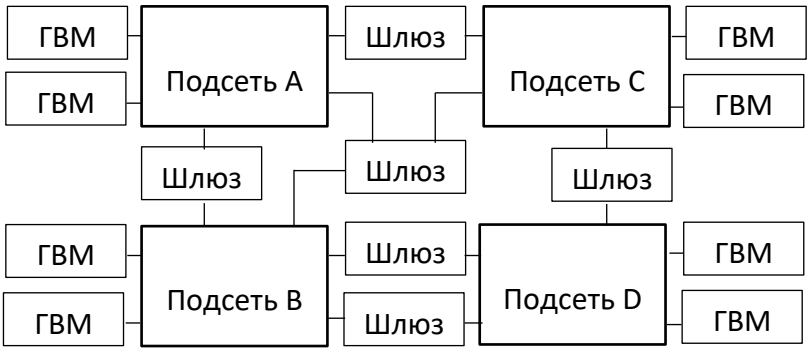


Рисунок 2.12 – Схема модели взаимного соединения подсетей

Далее межсетевая дейтаграмма в составе пакета этой новой подсети отправляется в адрес выбранного шлюза (или ГВМ).

Реализация IP не обеспечивает для пользователей объединенной сети архитектуры DARPA услуг типа виртуальных каналов. Такие услуги обеспечиваются средствами транспортного протокола TCP.

9.2.1 Межсетевой протокол IPv4

Протокол IP находится на межсетевом уровне стека протоколов TCP/IP. Функции протокола IP определены в стандарте RFC-791 следующим образом: «Протокол IP обеспечивает передачу блоков данных, называемых дейтаграммами, от отправителя к получателям, где отправители и получатели являются компьютерами, идентифицируемыми адресами фиксированной длины (*IP-адресами*). Протокол IP обеспечивает при необходимости также фрагментацию и сборку дейтаграмм для передачи данных через сети с малым размером пакетов».

Протокол IP является *ненадежным* протоколом *без установления соединения*. Это означает, что протокол IP не подтверждает доставку данных, не контролирует целостность полученных данных и не производит операцию квитирования (handshaking) – обмена служебными сообщениями, подтверждающими установку соединения с узлом назначения и его готовность к приему данных. Протокол IP обрабатывает каждую дейтаграмму как независимую единицу, не имеющую связи ни с какими другими дейтаграммами в Интернет. После того, как дейтаграмма отправляется в сеть, ее дальнейшая судьба никак не контролируется отправителем (на уровне протокола IP). Если дейтаграмма не может быть доставлена, она уничтожается. Узел, уничтоживший дейтаграмму, может оправить по обратному адресу *ICMP-сообщение* о причине сбоя.

Гарантию правильной передачи данных предоставляют протоколы вышестоящего уровня (например, протокол TCP), которые имеют для этого необходимые механизмы.

Формат заголовка IP-дейтаграммы. IP-дейтаграмма состоит из заголовка и данных. Заголовок дейтаграммы состоит из 32-разрядных слов и имеет переменную длину, зависящую от размера поля “Options”, но всегда кратную 32 битам. За заголовком непосредственно следуют данные, передаваемые в дейтаграмме. Формат заголовка приведен на рисунке 2.13.

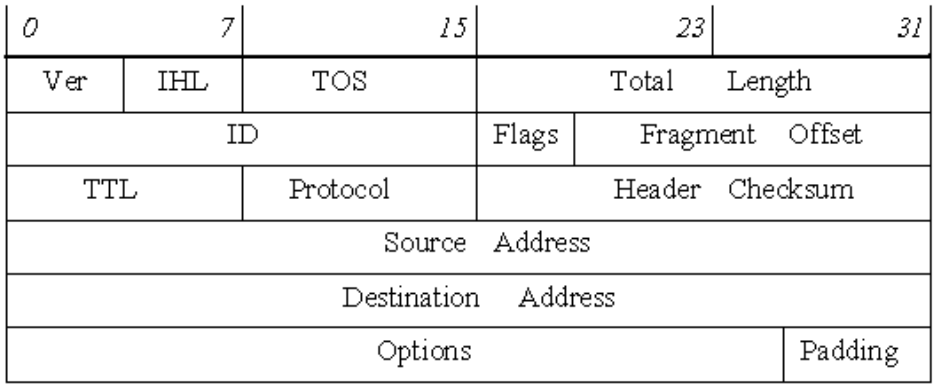


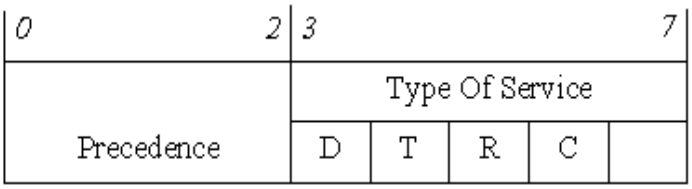
Рисунок 2.13 – Формат заголовка IP-дейтаграммы

Значения полей заголовка следующие.

Ver (4 бита) - версия протокола IP, в настоящий момент используется версия 4, новые разработки имеют номера версий 6-8.

IHL (Internet Header Length) (4 бита) - длина заголовка в 32-битных словах; диапазон допустимых значений от 5 (минимальная длина заголовка, поле “Options” отсутствует) до 15 (т.е. может быть максимум 40 байт опций).

TOS (Type Of Service) (8 бит) - значение поля определяет приоритет дейтаграммы и желаемый тип маршрутизации. Структура байта TOS:



Три младших бита (“Precedence”) определяют приоритет дейтаграммы:

- 111 – управление сетью;
- 110 – межсетевое управление;
- 101 – CRITIC-ECP;
- 100 – более чем мгновенно;
- 011 – мгновенно;
- 010 – немедленно;
- 001 – срочно;
- 000 – обычно.

Биты D, T, R, C определяют желаемый тип маршрутизации:

- D (Delay) - выбор маршрута с минимальной задержкой;
- T (Throughput) - выбор маршрута с максимальной пропускной способностью;
- R (Reliability) - выбор маршрута с максимальной надежностью;
- C (Cost) - выбор маршрута с минимальной стоимостью.

В дейтаграмме может быть установлен только один из битов D, T, R, C. Старший бит байта не используется.

Реальный учет приоритетов и выбора маршрута в соответствии со значением байта TOS зависит от маршрутизатора, его программного обеспечения и настроек. Маршрутизатор может поддерживать расчет маршрутов для всех типов TOS, для части или игнорировать TOS вообще. Маршрутизатор может учитывать значение приоритета при обработке всех дейтаграмм или при обработке дейтаграмм, исходящих только из некоторого ограниченного множества узлов сети, или вовсе игнорировать приоритет.

Total Length (16 бит) - длина всей дейтаграммы в октетах, включая заголовок и данные, максимальное значение 65535, минимальное - 21 (заголовок без опций и один октет в поле данных).

ID (Identification) (16 бит), *Flags* (3 бита), *Fragment Offset* (13 бит) используются для фрагментации и сборки дейтаграмм.

TTL (Time To Live) (8 бит) - “время жизни” дейтаграммы. Устанавливается отправителем, измеряется в секундах. Каждый маршрутизатор, через который проходит дейтаграмма, переписывает значение TTL, предварительно вычтя из него время, потраченное на обработку дейтаграммы. Так как в настоящее время скорость обработки данных на маршрутизаторах велика, на одну дейтаграмму тратится обычно меньше секунды, поэтому фактически каждый маршрутизатор вычитает из TTL единицу. При достижении значения TTL=0 дейтаграмма уничтожается, при этом отправителю может быть послано соответствующее ICMP-сообщение. Контроль TTL предотвращает зацикливание дейтаграммы в сети.

Protocol (8 бит) - определяет программу (вышестоящий протокол стека), которой должны быть переданы данные дейтаграммы для дальнейшей обработки. Коды некоторых протоколов приведены в таблице 2.2.

Header Checksum (16 бит) - контрольная сумма заголовка, представляет из себя 16 бит, дополняющие биты в сумме всех 16-битовых слов заголовка. Перед вычислением контрольной суммы значение поля “Header Checksum” обнуляется. Поскольку маршрутизаторы изменяют значения некоторых полей заголовка при обработке дейтаграммы (как минимум, поля “TTL”), контрольная сумма каждым маршрутизатором пересчитывается заново. Если при проверке контрольной суммы обнаруживается ошибка, дейтаграмма уничтожается.

Таблица 2.2 – Коды протоколов

Код	Протокол	Описание
1	ICMP	Протокол контрольных сообщений
2	IGMP	Протокол управления группой хостов
4	IP	IP поверх IP (инкапсуляция)
6	TCP	TCP
8	EGP	Протокол внешней маршрутизации (устарел)
9	IGP	Протокол внутренней маршрутизации (устарел)
17	UDP	UDP
46	RSVP	Протокол резервирования ресурсов при мультикастинге
88	IGRP	Протокол внутренней маршрутизации от фирмы CISCO
89	OSPF	Протокол внутренней маршрутизации

Source Address (32 бита) - IP-адрес отправителя.

Destination Address (32 бита) - IP-адрес получателя.

Options - опции, поле переменной длины. Опций может быть одна, несколько или ни одной. Опции определяют дополнительные услуги модуля IP по обработке дейтаграммы, в заголовок которой они включены.

Padding - выравнивание заголовка по границе 32-битного слова, если список опций занимает нецелое число 32-битных слов. Поле “Padding” заполняется нулями.

Фрагментация дейтаграмм. Различные среды передачи имеют различный максимальный размер передаваемого блока данных (MTU - Media Transmission Unit), это число зависит от скоростных характеристик среды и вероятности возникновения ошибки при передаче. Например, размер MTU в 10Мбит/с Ethernet равен 1536 октетам, в 100 Мбит/с FDDI - 4096 октетам.

При передаче дейтаграммы из среды с большим MTU в среду с меньшим MTU может возникнуть необходимость во фрагментации дейтаграммы. Фрагментация и сборка дейтаграмм осуществляются модулем протокола IP. Для этого применяются поля “ID” (Identification), “Flags” и “Fragment Offset” заголовка дейтаграммы.

Flags – поле состоит из 3 бит, младший из которых всегда сброшен:

0	DF	MF

Значения бита DF (Don’t Fragment): 0 - фрагментация разрешена, 1 - фрагментация запрещена (если дейтаграмму нельзя передать без фрагментации, она уничтожается).

Значения бита MF (More Fragments): 0 - данный фрагмент последний (единственный), 1 - данный фрагмент не последний.

ID (Identification) – идентификатор дейтаграммы, устанавливается отправителем; используется для сборки дейтаграммы из фрагментов для определения принадлежности фрагментов одной дейтаграмме.

Fragment Offset – смещение фрагмента, значение поля указывает, на какой позиции в поле данных исходной дейтаграммы находится данный фрагмент. Смещение считается 64-битовыми порциями, т.е. минимальный размер фрагмента равен 8 октетам, а следующий фрагмент в этом случае будет иметь смещение 1. Первый фрагмент имеет смещение нуль.

Опции IP. Опции определяют дополнительные услуги протокола IP по обработке дейтаграмм. Опция состоит, как минимум, из октета “Тип опции”, за которым могут следовать октет “Длина опции” и октеты с данными для опции.

Структура октета “Тип опции”:

0	*1 2*	*3 7*
C	класс	номер

Значения бита C:

1 - опция копируется во все фрагменты;

0 - опция копируется только в первый фрагмент.

Определены два класса опций: 0 - “Управление” и 2 - “Измерение и отладка”. Внутри класса опция идентифицируется номером. Ниже в таблице 2.3 приведены опции, описанные в стандарте протокола IP; знак “-” в столбце “Октет длины”

означает, что опция состоит только из октета “Тип опции”, число рядом с плюсом означает, что опция имеет фиксированную длину (длина указывается в октетах).

Таблица 2.3 – Опции IP

Класс	Номер	Октет длины	Опция
0	0	-	Конец списка опций
0	1	-	Нет операции
0	2	+ (11)	Безопасность
0	3	+	Loose Source Routing (свободное исполнение маршрута отправителя)
0	9	+	Strict Source Routing (строгое исполнение маршрута отправителя)
0	7	+	Запись маршрута
0	8	+ (4)	Stream ID
2	4	+	Internet Timestamp (временной штамп)

При обнаружении в списке опции “Конец списка опций” разбор опций прекращается, даже если длина заголовка (IHL) еще не исчерпана. Опция “Нет операции” обычно используется для выравнивания между опциями по границе 32 бит.

Большинство опций в настоящее время не используются. Опции “Stream ID” и “Безопасность” применялись в ограниченном круге экспериментов, функции опций “Запись маршрута” и “Internet Timestamp” выполняет программа traceroute. Определенный интерес представляют только опции “Loose/Strict Source Routing”, они рассмотрены в следующем пункте.

Применение опций в дейтаграммах замедляет их обработку. Поскольку большинство дейтаграмм не содержат опций, то есть имеют фиксированную длину заголовка, их обработка максимально оптимизирована именно для этого случая. Появление опции прерывает этот скоростной процесс и вызывает стандартный универсальный модуль IP, способный обработать любые стандартные опции, но за счет существенной потери в быстродействии.

Опции “Loose/Strict Source Routing”. Опции “Loose/Strict Source Routing” (класс 0, номера 3 и 9 соответственно) предназначены для указания дейтаграмме предопределенного отправителем маршрута следования.

Обе опции выглядят одинаково:

Тип опции	Длина опции	Указатель	Данные
1 октет	*1 октет*	*1 октет*	*(Длина - 3) октетов*

Поле “Данные” содержит список IP-адресов требуемого маршрута в порядке следования. Поле “Указатель” служит для определения очередного пункта маршрута, оно содержит номер первого октета IP-адреса этого пункта в поле “Данные”. Номера считаются от начала опции с единицы, начальное значение указателя - 4.

Опции работают следующим образом.

Предположим, дейтаграмма, посланная из A в B, должна проследовать через маршрутизаторы G1 и G2. На выходе из A поле “Destination Address” заголовка дейтаграммы содержит адрес G1, а поле данных опции - адреса G2 и B (указатель=4). По прибытии дейтаграммы в G1 из поля данных опции, начиная с октета, указанного указателем (октет 4), извлекается адрес следующего пункта (G2) и помещается в поле “Destination Address”, при этом значение указателя увеличивается на 4, а на место адреса G2 в поле данных опции помещается адрес того интерфейса маршрутизатора G1, через который дейтаграмма будет отправлена по новому месту назначения (то есть в G2). По прибытии дейтаграммы в G2 процедура повторяется и дейтаграмма отсылается в B. При обработке дейтаграммы в B обнаруживается, что значение указателя (12) превышает длину опции, это значит, что конечный пункт маршрута достигнут.

Отличия опций “Loose Source Routing” и “Strict Source Routing” друг от друга заключаются в следующем:

“Loose”: очередной пункт требуемого маршрута может быть достигнут за любое количество шагов (*хопов*);

“Strict”: очередной пункт требуемого маршрута должен быть достигнут за 1 шаг, то есть непосредственно.

Рассмотренные опции копируются во все фрагменты. В дейтаграмме может быть только одна такая опция.

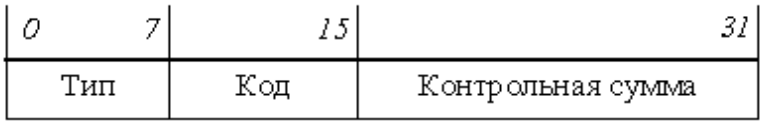
Опции “Loose/Strict Source Routing” могут быть использованы в целях несанкционированного проникновения через контролирующий (фильтрующий) узел (в поле “Destination Address” устанавливается разрешенный адрес, дейтаграмма пропускается контролирующим узлом, далее из поля данных опции подставляется запрещенный адрес и дейтаграмма перенаправляется по этому адресу уже за пределами досягаемости контролирующего узла), поэтому в целях безопасности рекомендуется вообще запретить пропуск контролирующим узлом дейтаграмм с рассматриваемыми опциями.

9.2.2 Протокол ICMP

Протокол ICMP (Internet Control Message Protocol, Протокол Управляющих Сообщений Интернет) является неотъемлемой частью IP-модуля. Он обеспечивает обратную связь в виде диагностических сообщений, посылаемых отправителю при невозможности доставки его дейтаграммы и в других случаях. ICMP стандартизован в RFC-792, дополнения — в RCF-950,1256.

ICMP-сообщения не порождаются при невозможности доставки: дейтаграмм, содержащих ICMP-сообщения; не первых фрагментов дейтаграмм; дейтаграмм, направленных по групповому адресу (широковещание, мультикастинг); дейтаграмм, адрес отправителя которых нулевой или групповой. Все ICMP-сообщения имеют IP-заголовок, значение поля “Protocol” равно 1. Данные дейтаграммы с ICMP-сообщением не передаются вверх по стеку протоколов для обработки, а обрабатываются IP-модулем.

После IP-заголовка следует 32-битное слово с полями “Тип”, “Код” и “Контрольная сумма”:



Поля типа и кода определяют содержание ICMP-сообщения. Формат остальной части дейтаграммы зависит от вида сообщения. Контрольная сумма считается так же, как и в IP-заголовке, но в этом случае суммируется содержимое ICMP-сообщения, включая поля “Тип” и “Код”. Виды ICMP-сообщений приведены в таблице 2.4.

Таблица 2.4 – Виды ICMP-сообщений

Тип	Сообщение
0	Echo Reply (эхо-ответ)
3	Destination Unreachable (адресат недостижим по различным причинам; причина указывается в поле Код)
4	Source Quench (замедление источника)
5	Redirect (выбрать другой маршрутизатор для посылки дейтаграмм)
8	Echo Request (эхо-запрос)
9	Router Advertisement (объявление маршрутизатора)
10	Router Solicitation (запрос объявления маршрутизатора)
11	Time Exceeded (время жизни дейтаграммы истекло)
12	Parameter problem (ошибка в параметрах)
13	Timestamp (запрос временной метки)
14	Timestamp Reply (ответ на запрос временной метки)
17	Address Mask Request (запрос сетевой маски)
18	Address Mask Reply (ответ на запрос сетевой маски)

9.2.3 IPv4-адресация

IP-адрес является уникальным 32-битным идентификатором IP-интерфейса в Интернет. Часто говорят, что IP-адрес присваивается узлу сети (например, хосту); это верно в случае, если узел является хостом с одним IP-интерфейсом, иначе следует уточнить, об адресе какого именно интерфейса данного узла идет речь. Далее для краткости там, где это не вызовет неверного толкования, вместо *адреса IP-интерфейса узла сети* говорится об *IP-адресе хоста*.

IP-адреса принято записывать разбивкой всего адреса по октетам, каждый октет записывается в виде десятичного числа, числа разделяются точками. Например, адрес

10100000010100010000010110000011

записывается как

10100000.01010001.00000101.10000011 = 160.81.5.131.

IP-адрес хоста состоит из номера IP-сети, который занимает старшую область адреса, и номера хоста в этой сети, который занимает младшую часть. Положение границы сетевой и хостовой частей (обычно оно характеризуется количеством

бит, отведенных на номер сети) может быть различным, определяя различные типы IP-адресов, которые рассматриваются ниже.

Классовая модель. В классовой модели IP-адрес может принадлежать к одному из четырех классов сетей. Каждый класс характеризуется определенным размером сетевой части адреса, кратным восьми; таким образом, граница между сетевой и хостовой частями IP-адреса в классовой модели всегда проходит по границе октета. Принадлежность к тому или иному классу определяется по старшим битам адреса (рисунок 2.14).

Класс A. Старший бит адреса равен нулю. Размер сетевой части равен 8 битам. Таким образом, может существовать всего примерно 2^7 сетей класса A, но каждая сеть обладает адресным пространством на 2^{24} хостов. Так как старший бит адреса нулевой, то все IP-адреса этого класса имеют значение старшего октета в диапазоне 0 — 127, который является также и номером сети.

Класс B. Два старших бита адреса равны 10. Размер сетевой части равен 16 битам. Таким образом, может существовать всего примерно 2^{14} сетей класса B, каждая сеть обладает адресным пространством на 2^{16} хостов. Значения старшего октета IP-адреса лежат в диапазоне 128 — 191, при этом номером сети являются два старших октета.

Класс C. Три старших бита адреса равны 110. Размер сетевой части равен 24 битам. Количество сетей класса C примерно 2^{21}, адресное пространство каждой сети рассчитано на 254 хоста. Значения старшего октета IP-адреса лежат в диапазоне 192 - 223, а номером сети являются три старших октета.

Класс D. Сети со значениями старшего октета IP-адреса 224 и выше. Зарезервированы для специальных целей. Некоторые адреса используются для мультикастинга - передачи дейтаграмм группе узлов сети, например:

224.0.0.1 - всем хостам данной сети;

224.0.0.2 - всем маршрутизаторам данной сети;

224.0.0.5 - всем OSPF-маршрутизаторам;

224.0.0.6 - всем выделенным (designated) OSPF-маршрутизаторам;

В классе A выделены две особые сети, их номера 0 и 127. Сеть 0 используется при маршрутизации как указание на маршрут по умолчанию (см. п. 2.3) и в других особых случаях.

IP-интерфейс с адресом в сети 127 используется для адресации узлом себя самого (*loopback, интерфейс обратной связи*). Интерфейс обратной связи не обязательно имеет адрес в сети 127 (особенно у маршрутизаторов), но если узел имеет IP-интерфейс с адресом 127.0.0.1, то это - интерфейс обратной связи.

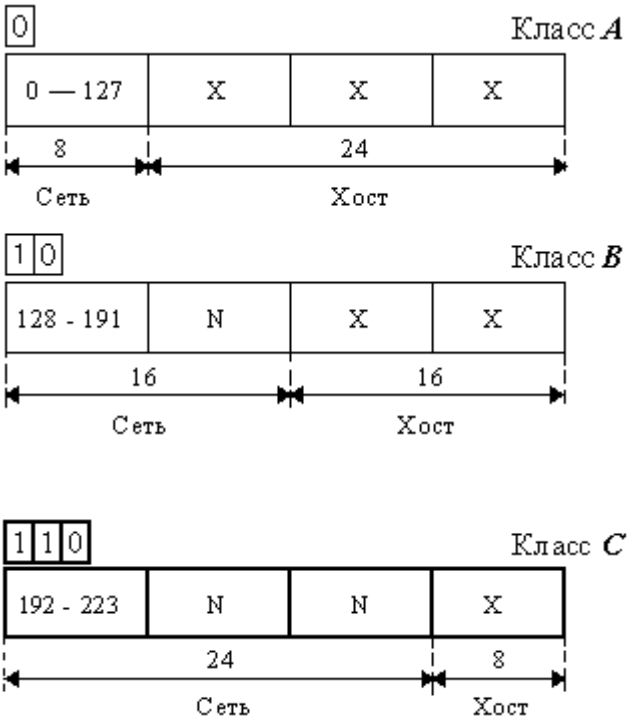


Рисунок 2.14 – Классы IP-адресов

Обращение по адресу loopback-интерфейса означает связь с самим собой (без выхода пакетов данных на уровень доступа к среде передачи); для протоколов на уровнях транспортном и выше такое соединение неотличимо от соединения с удаленным узлом, что удобно использовать, например, для тестирования сетевого программного обеспечения.

В любой сети (это справедливо и для бесклассовой модели, которую мы рассмотрим ниже) все нули в номере хоста обозначают саму сеть, все единицы - адрес широковещательной передачи (broadcast).

Например, 194.124.84.0 - сеть класса С, номер хоста в ней определяется последним октетом. При отправлении широковещательного сообщения оно отправляется по адресу 194.84.124.255. Номера, разрешенные для присваивания хостам: от 1 до 254 (194.84.124.1 — 194.84.124.254), всего 254 возможных адреса.

Другой пример: в сети 135.198.0.0 (класс В, номер хоста занимает два октета) широковещательный адрес 135.198.255.255, диапазон номеров хостов: 0.1 — 255.254 (135.198.0.1 — 135.198.255.254).

Бесклассовая модель (CIDR). Предположим, в локальной сети, подключаемой к Интернет, находится 2000 компьютеров. Каждому из них требуется выдать IP-адрес. Для получения необходимого адресного пространства нужны либо 8 сетей класса С, либо одна сеть класса В. Сеть класса В вмещает 65534 адреса, что много больше требуемого количества. При общем дефиците IP-адресов такое использование сетей класса В расточительно. Однако если мы будем использовать 8 сетей класса С, возникнет следующая проблема: каждая такая IP-сеть должна быть представлена отдельной строкой в таблицах маршрутов на

маршрутизаторах, потому что с точки зрения маршрутизаторов — это 8 абсолютно никак не связанных между собой сетей, маршрутизация дейтаграмм в которые осуществляется независимо, хотя фактически эти IP-сети и расположены в одной физической локальной сети и маршруты к ним идентичны. Таким образом, экономя адресное пространство, мы многократно увеличиваем служебный трафик в сети и затраты по поддержанию и обработке маршрутных таблиц.

С другой стороны, нет никаких формальных причин проводить границу сеть-хост в IP-адресе именно по границе октета. Это было сделано исключительно для удобства представления IP-адресов и разбиения их на классы. Если выбрать длину сетевой части в 21 бит, а на номер хоста отвести, соответственно, 11 бит, мы получим сеть, адресное пространство которой содержит 2046 IP-адресов, что максимально точно соответствует поставленному требованию. Это будет *одна* сеть, определяемая своим уникальным 21-битным номером, следовательно, для ее обслуживания потребуется только *одна* запись в таблице маршрутов.

Единственная проблема, которую осталось решить: как определить, что на сетевую часть отведен 21 бит? В случае классовой модели старшие биты IP-адреса определяли принадлежность этого адреса к тому или иному классу и, следовательно, количество бит, отведенных на номер сети.

В случае адресации вне классов, с произвольным положением границы сеть-хост внутри IP-адреса, к IP-адресу прилагается 32-битовая маска, которую называют *маской сети* (netmask) или *маской подсети* (subnet mask). Сетевая маска конструируется по следующему правилу:

- на позициях, соответствующих номеру сети, биты установлены;
- на позициях, соответствующих номеру хоста, биты сброшены.

Описанная выше модель адресации называется бесклассовой (CIDR - Classless Internet Direct Routing, прямая бесклассовая маршрутизация в Интернет). В настоящее время классовая модель считается устаревшей и маршрутизация и (большей частью) выдача блоков IP-адресов осуществляются по модели CIDR, хотя классы сетей еще прочно удерживаются в терминологии.

Запись адресов в бесклассовой модели. Для удобства записи IP-адрес в модели CIDR часто представляется в виде a.b.c.d / n, где a.b.c.d — IP адрес, n — количество бит в сетевой части.

Пример: 137.158.128.0/17.

Маска сети для этого адреса: 17 единиц (сетевая часть), за ними 15 нулей (хостовая часть), что в октетном представлении равно

11111111.11111111.10000000.00000000 = 255.255.128.0.

Представив IP-адрес в двоичном виде и побитно умножив его на маску сети, мы получим номер сети (все нули в хостовой части). Номер хоста в этой сети мы можем получить, побитно умножив IP-адрес на инвертированную маску сети.

Пример: IP = 205.37.193.134/26 или, что то же,

IP = 205.37.193.134 netmask = 255.255.255.192.

Распишем в двоичном виде:

IP = 11001101 00100101 11000111 10000110

маска = 11111111 11111111 11111111 11000000

Умножив побитно, получаем номер сети (в хостовой части - нули):

network = 11001101 00100101 11000111 10000000

или, в октетном представлении, 205.37.193.128/26, или, что то же, 205.37.193.128 netmask 255.255.255.192.

Хостовая часть рассматриваемого IP адреса равна 000110, или 6. Таким образом, 205.37.193.134/26 адресует хост номер 6 в сети 205.37.193.128/26. В классовой модели адрес 205.37.193.134 определял бы хост 134 в сети класса C 205.37.193.0, однако указание маски сети (или количества бит в сетевой части) однозначно определяет принадлежность адреса к бесклассовой модели.

Упражнение. Покажите, что адрес 132.90.132.5 netmask 255.255.240.0 определяет хост 4.5 в сети 132.90.128.0/20 (в классовой модели это был бы хост 132.5 в сети класса B 132.90.0.0). Найдите адрес broadcast для этой сети.

Очевидно, что сети классов A, B, C в бесклассовой модели представляются при помощи масок, соответственно, 255.0.0.0 (или /8), 255.255.0.0 (или /16) и 255.255.255.0 (или /24).

8.2.4 Маршрутизация IP-дейтаграмм

Процесс маршрутизации дейтаграмм состоит в определении следующего узла (*next hop*) в пути следования дейтаграммы и пересылки дейтаграммы этому узлу, который является либо узлом назначения, либо промежуточным маршрутизатором, задача которого — определить следующий узел и переслать ему дейтаграмму. Ни узел-отправитель, ни любой промежуточный маршрутизатор не имеют информации о всей цепочке, по которой пересылается дейтаграмма; каждый маршрутизатор, а также узел-отправитель, основываясь на адресе назначения дейтаграммы, находит только следующий узел ее маршрута.

Маршрутизация дейтаграмм осуществляется на уровне протокола IP.

Маршрутизация выполняется на основе данных, содержащихся в таблице маршрутов. Строка в таблице маршрутов состоит из следующих полей:

- адрес сети назначения;
- адрес следующего маршрутизатора (то есть узла, который знает, куда дальше отправить дейтаграмму, адресованную в сеть назначения);
- вспомогательные поля.

Таблица может составляться вручную или с помощью специализированных протоколов. Каждый узел сети, в том числе и хост, имеет таблицу маршрутов, хотя бы самую простую.

Пример маршрутизации. Рассмотрим процесс маршрутизации на примере.

Допустим (см. рисунок 2.15), хосты A и B находятся в сети 1, сеть 1 соединяется с сетью 2 с помощью маршрутизатора G1. К сети 2 подключен маршрутизатор G2, соединяющий ее с сетью 3, в которой находится хост C.

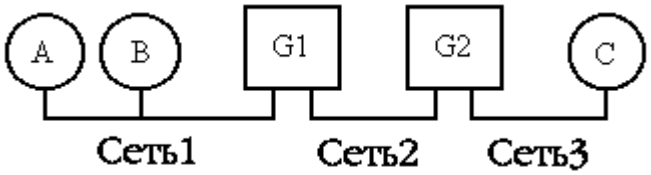


Рисунок 2.15 – Пример маршрутизации

Таблица маршрутов хоста A выглядит, например, так:

Сеть 1	A
Прочие сети	G1

Это означает, что дейтаграммы, адресованные узлам сети 1, отправляет сам хост A (так как это его локальная сеть), а дейтаграммы, адресованные в любую другую сеть (это называется маршрут по умолчанию), хост A отправляет маршрутизатору G1, чтобы тот занялся их дальнейшей судьбой.

Предположим, хост A посылает дейтаграмму хосту B. В этом случае, поскольку адрес B принадлежит той же сети, что и A, из таблицы маршрутов хоста A определяется, что доставка осуществляется непосредственно самим хостом A.

Если хост A отправляет дейтаграмму хосту C, то он определяет по IP-адресу C, что хост C не принадлежит к сети 1. Согласно таблице маршрутов A, все дейтаграммы с пунктами назначения, не принадлежащими сети 1, отправляются на маршрутизатор G1 (это называется *маршрут по умолчанию*). При этом хост A не знает, что маршрутизатор G1 будет делать с его дейтаграммой и каков будет ее дальнейший маршрут - это забота исключительно G1. G1 в свою очередь по своей таблице маршрутов определяет, что все дейтаграммы, адресованные в сеть 3, должны быть пересланы на маршрутизатор G2. Это может быть как явно указано в таблице, находящейся на G1, в виде

Сеть 3	G2,

так и указано в виде маршрута по умолчанию.

На этом функции G1 заканчиваются, дальнейший путь дейтаграммы ему неизвестен и его не интересует. Маршрутизатор G2, получив дейтаграмму, определяет, что она адресована в одну из сетей (№3), к которым он присоединен непосредственно, и доставляет дейтаграмму на хост C.

9.2.5 Пример подключения локальной сети организации к Интернет

Рассмотрим реальный пример подключения к Интернет локальной сети организации (рисунок 2.16). IP-адрес локальной сети - 194.84.124.0/24 (сеть класса C). В эту сеть включен маршрутизатор G1. IP-интерфейс этого маршрутизатора, подключенный к локальной сети Ethernet, имеет адрес 194.84.124.1. Второй IP-интерфейс маршрутизатора подключен к выделенной линии (синхронный последовательный канал), ведущей к провайдеру Интернет. К другому концу этой линии подключен IP-интерфейс маршрутизатора G2, принадлежащего провайдеру. Эти два интерфейса образуют отдельную сеть 194.84.0.116/30. В этой сети на номер интерфейса отведено всего 2 бита — 4 варианта адресов, из которых

один (00) обозначает саму сеть, один (11) — широковещательный; таким образом, в подобной сети может находиться всего 2 узла — это минимальная возможная сеть. Интерфейс маршрутизатора G1 в сети 194.84.0.116/30 имеет адрес 194.84.0.117, а маршрутизатора G2 — 194.84.0.118. Маршрутизатор G2 имеет еще некоторое количество интерфейсов, к части которых подключены выделенные линии от других клиентов, к части — локальные сети коммуникационного центра, другие маршрутизаторы и магистральные линии дальней связи.

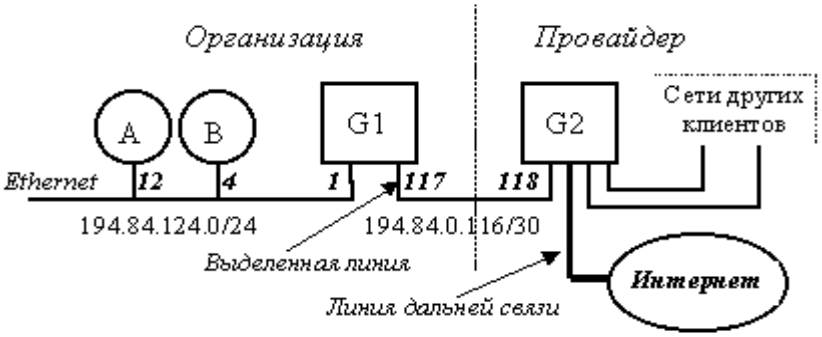


Рисунок 2.16 – Подключение локальной сети к Интернет

Маршрутизатор или шлюз? Небольшое замечание о терминологии. С точки зрения топологии соединений G2 (рисунок 2.16) является собственно *маршрутизатором (router)*, так как он коммутирует потоки дейтаграмм между своими многочисленными IP-интерфейсами, в то время как G1 является *шлюзом (gateway)* — интерфейсом между двумя разнородными средами передачи данных (локальной сетью Ethernet и выделенной линией). Все дейтаграммы, идущие вовне локальной сети, он безусловно транслирует на G2, и только G2 приступает именно к маршрутизации. Однако, с точки зрения общего подхода к задаче маршрутизации как определения *следующего маршрутизатора* в пути дейтаграммы на основе записей в таблице маршрутов, функции G1 и G2 не различаются, различается только сложность их маршрутных таблиц. Поэтому мы будем считать термины “маршрутизатор” (“router”) и “шлюз” (“gateway”) синонимами и будем использовать термин “шлюз”, говоря о маршрутизаторе, расположенном между локальной сетью конечного пользователя (например, сетью предприятия) и “внешним миром”.

Таблицы маршрутов. Рассмотрим примеры маршрутных таблиц, с которыми имеет дело администратор локальной сети 194.84.124.0/24.

Таблица маршрутов рядового хоста с адресом 194.84.124.4 (хост B на рисунке 2.16) приведена ниже (таблица 2.5).

Таблица 2.5 – Таблица маршрутов рядового хоста с адресом 194.84.124.4

Destination	Gateway	Flags	Interface
127.0.0.1	127.0.0.1	UH	lo0
194.84.124.0	194.84.124.4	U	le0
0.0.0.0	194.84.124.1	UG	

Значения флагов: U (Up) - маршрут работает; H (Host) - пунктом назначения является отдельный узел (хост), а не сеть; G (Gateway) - маршрут к сети назначения проходит через один или несколько промежуточных маршрутизаторов. Интерфейс le0 обозначает Ethernet, lo0 - интерфейс обратной связи (loopback).

Значение первой записи очевидно, вторая запись определяет, что дейтаграммы, адресованные в локальную сеть, хост отправляет самостоятельно через свой интерфейс le0. Третья запись (маршрут по умолчанию) устанавливает, что все остальные дейтаграммы передаются на адрес 194.84.124.1, который является адресом следующего маршрутизатора (флаг G), для дальнейшей пересылки. Чтобы определить способ достижения самого маршрутизатора, следует, очевидно, обратиться ко второй строке таблицы, так как адрес маршрутизатора принадлежит сети 194.84.124.0.

Заметим, что в этой таблице для простоты опущены маски сетей.

Пример таблицы маршрутов маршрутизатора, соединяющего локальную сеть с провайдером Интернет по выделенному каналу (G1 на рисунке 2.16) приведен в таблице 2.6.

Таблица 2.6 - Пример таблицы маршрутов маршрутизатора G1

Destination	Mask	Gateway	Interface
194.84.124.0	255.255.255.0	194.84.124.1	le0
194.84.0.116	255.255.255.252	194.84.0.117	se0
0.0.0.0	0.0.0.0	194.84.0.118	

В таблице явно показаны маски сетей. Первые две записи говорят о том, что маршрутизатор самостоятельно, через свои соответствующие IP-интерфейсы отправляет дейтаграммы, адресованные в сети, к которым он подключен непосредственно. Все остальные дейтаграммы перенаправляются к G2 (194.84.0.118). Интерфейс se0 обозначает последовательный (serial) канал - выделенную линию.

Создание статических маршрутов. Таблица маршрутов может заполняться различными способами. Статическая маршрутизация применяется в том случае, когда используемые маршруты не могут измениться в течение времени, например, для выше обсужденных хоста и маршрутизатора, где просто отсутствуют какие-либо альтернативные маршруты. Статические маршруты конфигурируются администратором сети или конкретного узла.

Для рядового хоста из рассмотренного выше примера достаточно указать только адрес шлюза (следующего маршрутизатора в маршруте по умолчанию), остальные записи в таблице очевидны, и хост, зная свой собственный IP-адрес и сетевую маску, может внести их самостоятельно. Адрес шлюза может быть указан как вручную, так и получен автоматически при конфигурировании стека TCP/IP через DHCP сервер.

Динамическая маршрутизация. В случае объединения сетей со сложной топологией, когда существует несколько вариантов маршрутов от одного узла к другому и (или) когда состояние сетей (топология, качество каналов связи) изменяется с течением времени, таблицы маршрутов составляются динамически с помощью различных протоколов маршрутизации. Подчеркнем, что протоколы маршрутизации не осуществляют собственно маршрутизацию дейтаграмм - она в любом случае производится модулем IP согласно записям в таблице маршрутов, как обсуждалось выше. Протоколы маршрутизации на основании тех или иных алгоритмов динамически редактируют таблицу маршрутов, то есть вносят и удаляют записи, при этом часть записей может по-прежнему статически вноситься администратором.

В зависимости от алгоритма работы различают *дистанционно-векторные* протоколы (distance vector protocols) и протоколы *состояния связей* (link state protocols).

По области применения существует разделение на протоколы *внешней* (exterior) и *внутренней* (interior) маршрутизации.

Дистанционно-векторные протоколы реализуют алгоритм Беллмана-Форда (Bellman-Ford). Общая схема их работы такова: каждый маршрутизатор периодически широковещательно рассылает информацию о расстоянии от себя до всех известных ему сетей (*"вектор расстояний"*). В начальный момент времени, разумеется, рассылается информация только о тех сетях, к которым маршрутизатор подключен непосредственно.

Также каждый маршрутизатор, получив от кого-либо вектор расстояний, в соответствии с полученной информацией корректирует уже имеющиеся у него данные о достижимости сетей или добавляет новые, указывая маршрутизатор, от которого получен вектор, в качестве *следующего маршрутизатора* на пути в данные сети. Через некоторое время алгоритм сходится и все маршрутизаторы имеют информацию о маршрутах до всех сетей.

Дистанционно-векторные протоколы хорошо работают только в небольших сетях. Развитие технологии векторов расстояний - "векторы путей", применяющиеся в протоколе BGP.

При работе *протоколов состояния связей* каждый маршрутизатор контролирует состояние своих связей с соседями и при изменении состояния (например, при обрыве связи) рассылает широковещательное сообщение, после получения которого все остальные маршрутизаторы корректируют свои базы данных и пересчитывают маршруты. В отличие от дистанционно-векторных протоколов протоколы состояния связей создают на каждом маршрутизаторе базу данных, описывающую полный граф сети и позволяющую локально и, следовательно, быстро производить расчет маршрутов.

Распространенный протокол такого типа, *OSPF*, базируется на алгоритме SPF (Shortest Path First) поиска кратчайшего пути в графе, предложенном Дейкстрой (E.W.Dijkstra).

Протоколы состояния связей существенно сложнее дистанционно-векторных, но обеспечивают более быстрое, оптимальное и корректное вычисление маршрутов.

Протоколы внутренней маршрутизации (например, RIP, OSPF; собирательное название IGP – Interior Gateway Protocols) применяются на маршрутизаторах, действующих внутри *автономных систем*. Автономная система – это наиболее крупное деление Интернет, представляющее из себя объединение сетей с одинаковой маршрутизационной политикой и общей администрацией.

Область действия того или иного протокола внутренней маршрутизации может охватывать не всю автономную систему, а только некоторое объединение сетей, являющееся частью автономной системы. Такое объединение мы будем называть *системой сетей*, или просто *системой*, иногда с указанием протокола маршрутизации, действующего в этой системе, например: RIP-система, OSPF-система.

Маршрутизация *между* автономными системами осуществляется *пограничными* (border) маршрутизаторами, таблицы маршрутов которых составляются с помощью протоколов внешней маршрутизации (собирательное название EGP – Exterior Gateway Protocols). Особенность протоколов внешней маршрутизации состоит в том, что при расчете маршрутов они должны учитывать не только топологию графа сети, но и политические ограничения, вводимые администрацией автономных систем на маршрутизацию через свои сети трафика других автономных систем. В настоящее время наиболее распространенным протоколом внешней маршрутизации является BGP.

Протокол ARP. Протокол ARP (Address Resolution Protocol, Протокол распознавания адреса) предназначен для преобразования IP-адресов в MAC-адреса, часто называемые также физическими адресами.

MAC расшифровывается как Media Access Control, контроль доступа к среде передачи. MAC-адреса идентифицируют устройства, подключенные к физическому каналу, пример MAC-адреса – адрес Ethernet.

Для передачи IP-дейтаграммы по физическому каналу (будем рассматривать Ethernet) требуется инкапсулировать эту дейтаграмму в кадр Ethernet и в заголовке кадра указать адрес Ethernet-карты, на которую будет доставлена эта дейтаграмма для ее последующей обработки вышестоящим по стеку протоколом IP. IP-адрес, включенный в заголовок дейтаграммы, адресует IP-интерфейс какого-либо узла сети и не заключает в себе никаких указаний ни на физическую среду передачи, к которой подключен этот интерфейс, ни тем более на физический адрес устройства (если таковой имеется), с помощью которого этот интерфейс сообщается со средой.

Поиск по данному IP-адресу соответствующего Ethernet-адреса производится протоколом ARP, функционирующим на уровне доступа к среде передачи. Протокол поддерживает в оперативной памяти динамическую arp-таблицу в целях

кэширования полученной информации. Порядок функционирования протокола следующий.

С межсетевого уровня поступает IP-дейтаграмма для передачи в физический канал (Ethernet), вместе с дейтаграммой передается, среди прочих параметров, IP-адрес узла назначения. Если в arp-таблице не содержится записи об Ethernet-адресе, соответствующем нужному IP-адресу, модуль arp ставит дейтаграмму в очередь и формирует широковещательный запрос. Запрос получают все узлы, подключенные к данной сети; узел, опознавший свой IP-адрес, отправляет arp-ответ (arp-response) со значением своего адреса Ethernet. Полученные данные заносятся в таблицу, ждущая дейтаграмма извлекается из очереди и передается на инкапсуляцию в кадр Ethernet для последующей отправки по физическому каналу.

ARP-запрос или ответ включается в кадр Ethernet непосредственно после заголовка кадра.

Форматы запроса и ответа одинаковы (рисунок 2.17) и отличаются только кодом операции (Operation code, 1 и 2 соответственно).

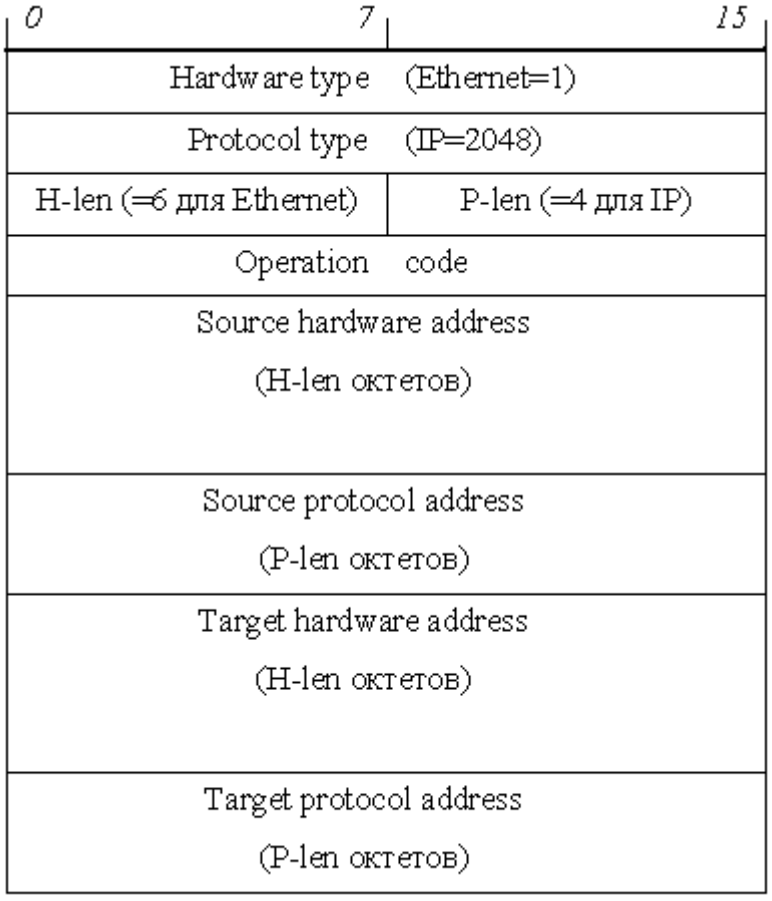


Рисунок 2.17 – Форматы ARP запроса и ответа

Несмотря на то, что ARP создавался специально для Ethernet, этот протокол может поддерживать различные типы физических сред (поле “Hardware type (Тип физической среды)”, значение 1 соответствует Ethernet), а также различные типы обслуживаемых протоколов (поле “Protocol type (Тип протокола)”, значение 2048 соответствует IP). Поля H-len и P-len содержат длины физического и

“протокольного” адресов соответственно, в октетах. Для Ethernet H-len=6, для IP P-len=4.

Поля “Source hardware address” и “Source protocol address” содержат физический (Ethernet) и “протокольный” (IP) адреса отправителя. Поля “Target hardware address” и “Target protocol address” содержат соответствующие адреса получателя. При посылке запроса поле “Target hardware address” инициализируется нулями, а в поле “Destination (Адрес назначения)” заголовка Ethernet-кадра ставится широковещательный адрес.

ARP для дейтаграмм, направленных в другую сеть. Дейтаграмма, направленная во внешнюю (в другую) сеть, должна быть передана маршрутизатору. Предположим, хост A отправляет дейтаграмму хосту B через маршрутизатор G. Несмотря на то, что в заголовке дейтаграммы, отправляемой из A, в поле “Destination” указан IP-адрес B, кадр Ethernet, содержащий эту дейтаграмму, должен быть доставлен маршрутизатору. Это достигается тем, что IP-модуль при вызове ARP-модуля передает тому вместе с дейтаграммой в качестве IP-адреса узла назначения адрес маршрутизатора, извлеченный из таблицы маршрутов. Таким образом, дейтаграмма с адресом B инкапсулируется в кадр с MAC-адресом G:

IP Source: A IP Destination: B	Заголовок дейтаграммы
Ethernet Source: A Ethernet Destination: G	Заголовок кадра Ethernet

Модуль Ethernet на маршрутизаторе G получает из сети этот кадр, так как кадр адресован ему, извлекает из кадра данные (то есть дейтаграмму) и отправляет их для обработки модулю IP. Модуль IP обнаруживает, что дейтаграмма адресована не ему, а хосту B, и по своей таблице маршрутов определяет, куда ее следует переслать. Далее дейтаграмма опять опускается на нижний уровень, к соответствующему физическому интерфейсу, которому передается в качестве IP-адреса узла назначения адрес следующего маршрутизатора, извлеченный из таблицы маршрутов, или сразу адрес хоста B, если маршрутизатор G может доставить дейтаграмму непосредственно к нему.

Proxy ARP. ARP-ответ может отправляться не обязательно искомым узлом, вместо него это может сделать другой узел. Такой механизм называется *proxy ARP*.

Рассмотрим пример (рисунок 2.18). Удаленный хост A подключается по коммутируемой линии к сети 194.84.124.0/24 через сервер доступа G. Сеть 194.84.124.0 на физическом уровне представляет собой Ethernet. Сервер G выдает хосту A IP-адрес 194.84.124.30, принадлежащий сети 194.84.124.0. Следовательно,

любой узел этой сети, например, хост B, полагает, что может непосредственно отправить дейтаграмму хосту A, поскольку они находятся в одной IP-сети.

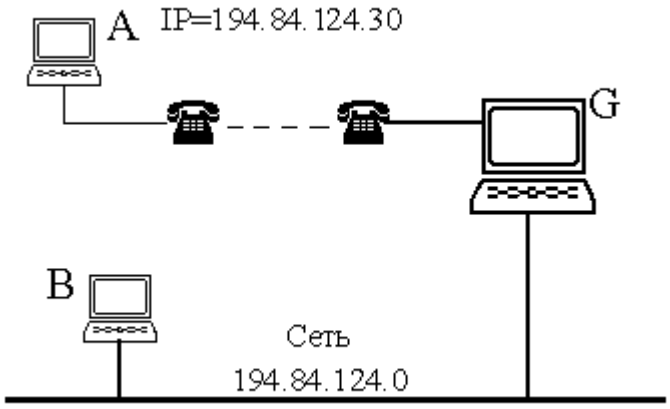


Рисунок 2.18 – Proxy ARP

IP-модуль хоста B вызывает ARP-модуль для определения физического адреса A. Однако вместо A (который, разумеется, откликнуться не может, потому что физически не подключен к сети Ethernet) откликается сервер G, который и возвращает свой Ethernet-адрес как физический адрес хоста A. Вслед за этим B отправляет, а G получает кадр, содержащий дейтаграмму для A, которую G отправляет адресату по коммутируемому каналу.

10 Транспортные протоколы

Рассмотренные ранее протоколы трех нижних уровней Эталонной модели взаимодействия открытых систем обеспечивают функционирование сети. передачи данных. Естественно, что эти протоколы в той или иной степени зависят от структуры сети передачи данных. В свою очередь, протоколы верхних уровней, начиная с сеансового уровня, также не осуществляют управление обменом (передачей) данных между процессами. Реализация этих функций возлагается на транспортную службу и транспортные протоколы, основным назначением которых и является обеспечение надежной связи между прикладными процессами компьютерной сети. Соответственно, в Эталонной модели взаимодействия открытых систем протоколы транспортного уровня занимают промежуточное положение между сетевыми и прикладными протоколами.

Как известно, качество передачи информации во многом определяется используемой сетью связи. Естественно, что при допустимом уровне надежности передачи, например, при оптоволоконных линиях связи, достаточно использовать минимальный набор транспортных услуг и простейший протокол обмена информацией. Особое значение транспортные протоколы приобретают в компьютерных сетях, передающая среда которых характеризуется относительно высоким уровнем ошибок и низкой надежностью передачи данных. В данном

случае обеспечение требуемого уровня качества передачи информации достигается за счет усложнения протокола транспортного уровня.

Одним из первых протоколов транспортного уровня является протокол AHHP (ARPA Host-to-Host Protocol), разработанный для сети ARPA. Учитывая, что первоначально эта сеть обеспечивала достаточно надежную передачу сообщений, основное внимание в протоколе AHHP уделялось управлению потоком данных, адресации пользователей, а также взаимодействию с программами, реализующими протоколы верхних Уровней. Развитие сети ARPA в направлении использования сетей передачи данных общего пользования привело к появлению нового, более надежного протокола, известного в настоящее время под названием "протокол управления передачей" или TCP (сокращение от Transmission Control Protocol). Главным образом этот протокол используется в компьютерных сетях с коммутацией пакетов. Протокол TCP оказался достаточно Удачным и был положен в основу стандартного международного протокола транспортного уровня. Соответственно, МККТТ определил рекомендацию X.224 для данного транспортного протокола, а также рекомендацию X.214 для транспортной службы.

С целью выбора оптимального набора транспортных услуг стандартным протоколом определено три типа (A, B, C) сетевых соединений и пять классов (0, 1, 2, 3, 4) транспортного протокола. В зависимости от характеристик конкретной сети передачи данных определяется тип сетевого соединения, которому она удовлетворяет. Затем, с учетом требуемого уровня качества передачи выбирается необходимый класс транспортного протокола.

Рассмотрим более подробно соотношение типов сетевых соединений и классов транспортных протоколов. Так, тип А определяет сетевое соединение с приемлемой частотой ошибок и допустимой интенсивностью сбоев, о появлении которых сообщается транспортной службе. Считается, что пакеты не теряются, и не нарушается порядок их следования. В этом случае на транспортном уровне нет необходимости предусматривать услуги восстановления сбоев, информирования о потере данных, восстановления последовательности и т. д.

Сетевые соединения типа В определяются как соединения с призмой частотой ошибок, но неприемлемой интенсивностью сигнализируемых повреждений. В этом случае транспортный протокол сам должен поддерживать режим восстановления при сбоях.

Наконец, сетевые соединения класса С — это такие соединения, которых частота сбоев неприемлема для пользователя транспортной сети. В этом случае транспортный протокол должен обладать возможностью обнаружения сетевых сбоев и восстановления соединения, а также определять нарушения в порядке следования пакетов и восстанавливать его.

Выбор класса определяется качеством обслуживания, которое запрашивает пользователь транспортной услуги, а также нижестоящим сетевым соединением (или соединениями), предоставляющим необходимые услуги.

Так 0 класс (TPO) определяет простейший тип транспортного соединения, для которого определен минимум функций. Для этого класса транспортного

протокола требуется только установить простое транспортное соединение. При сбоях он не предусматривает восстановление транспортных соединений. Предполагается, что сетевое соединение обеспечивает приемлемый уровень ошибок, а также допустимую частоту сетевых сбоев, о каждом из которых сообщается на вышестоящие уровни. |Подобные условия выполняются в рамках сетевого соединения типа А, которое и рассчитан 0 класс транспортного соединения.

Класс 1 (ТР1) — это тоже простой класс транспортного соединения по сравнению с 0 классом в него включены основные возможности восстановления при сбоях. Сбои могут происходить по ряду причин, в частности из-за разъединения или повреждений в сети, а также в случае приема блока данных неопознанного транспортного соединения.

Классы 2, 3 и 4 (ТР2, ТРЗ и ТР4) — это соответственно более сложные классы, предоставляющие дополнительные услуги по повышению надежности сетевого соединения. Эти услуги реализуются с помощью специальных управляющих процедур и примитивов. *Примитивы являются абстрактными* представлениями взаимодействий через точки доступа услуг, указывающими, что между пользователем и поставщиком услуги передается информация. Пользователь транспортной услуги и объект находятся на одном и том же уровне и взаимодействуют между собой путем обмена примитивами транспортных услуг. В свою очередь транспортный объект и поставщик сетевой услуги взаимодействует путем обмена примитивами сетевых услуг через межуровневый интерфейс.

Транспортный протокол не регламентирует, как должны быть реализованы примитивы, определяется лишь их состав и выполняемые функции. Предписаны четыре типа примитивов: *запрос, признак, ответ* и подтверждение. Требуемый уровень надежности передачи обеспечивается рядом транспортных функций, основными среди которых являются:

- создание соединений между портами процессов (сами порты создаются на сеансовом уровне);
- передача сообщений через установленные соединения;
- обнаружение сбоев и восстановление;
- обнаружение дубликатов пакетов;
- упорядочение передачи пакетов (при дейтаграммной передаче);
- фрагментация — разбивка сообщений на пакеты оптимальной длины;
- управление потоками и буферизация информации;
- синхронизация передачи информации (например, при передаче речи);
- организация приоритетных передач пакетов;
- защита передачи данных;
- инициализация и восстановление из состояния отказа.

10.1 Базовые характеристики транспортных протоколов TCP, UDP

10.1.1 Протокол управления передачей TCP

Протокол TCP (Transmission Control Protocol) обеспечивает сквозную доставку данных между прикладными процессами, запущенными на узлах, взаимодействующих по сети. Стандартное описание TCP содержится в RFC-793.

TCP - *надежный* байт-ориентированный (byte-stream) протокол *с установлением соединения*. TCP находится на транспортном уровне стека TCP/IP. Протокол IP занимается пересылкой дейтаграмм по сети, никак не гарантируя доставку, целостность, порядок прибытия информации и готовность получателя к приему данных; все эти задачи возложены на протокол TCP.

При получении дейтаграммы, в поле Protocol которой указан код протокола TCP (6), модуль IP передает данные этой дейтаграммы модулю TCP. Эти данные представляют собой TCP-сегмент, содержащий TCP-заголовок и данные пользователя (прикладного процесса). Модуль TCP анализирует служебную информацию заголовка, определяет, какому именно процессу предназначены данные пользователя, проверяет целостность и порядок прихода данных и подтверждает их прием другой стороне. По мере получения правильной последовательности неискаженных данных пользователя они передаются прикладному процессу.

Базовая передача данных. Модуль TCP выполняет передачу непрерывных потоков данных между своими клиентами в обоих направлениях. Клиентами TCP являются прикладные процессы, вызывающие модуль TCP при необходимости получить или отправить данные процессу-клиенту на другом узле. Протокол TCP рассматривает данные клиента как непрерывный неинтерпретируемый поток октетов. TCP разделяет этот поток на части для пересылки на другой узел в TCP-сегментах некоторого размера. Для отправки или получения сегмента модуль TCP вызывает модуль IP.

Немедленное отправление данных может быть затребовано процессом-клиентом от TCP-модуля с помощью специальной функции PUSH, иначе TCP сам будет решать, как накапливать и когда отправлять данные клиента или когда передавать клиенту полученные данные.

Обеспечение достоверности передачи. Модуль TCP обеспечивает защиту от повреждения, потери, дублирования и нарушения очередности получения данных. Для выполнения этих задач все октеты в потоке данных сквозным образом пронумерованы в возрастающем порядке. Заголовок каждого сегмента содержит число октетов данных в сегменте и порядковый номер первого октета той части потока данных, которая пересылается в данном сегменте. Например, если в сегменте пересылаются октеты с номерами от 2001 до 3000, то номер первого октета в данном сегменте равен 2001, а число октетов равно 1000. Номер первого байта в потоке определяется на этапе установления соединения и обозначается

ISN+. Также для каждого сегмента вычисляется контрольная сумма, позволяющая обнаружить повреждение данных.

При удачном приеме октета данных принимающий модуль посылает отправителю подтверждение о приеме - номер удачно принятого октета. Если в течение некоторого времени отправитель не получит подтверждения, считается, что октет не дошел или был поврежден, и он посылается снова. Этот механизм контроля надежности называется PAR (Positive Acknowledgment with Retransmission). В действительности подтверждение посылается не для одного октета, а для некоторого числа последовательных октетов.

Нумерация октетов используется также для упорядочения данных в порядке очередности и обнаружения дубликатов (которые могут быть посланы из-за большой задержки при передаче подтверждения или потери подтверждения).

Адресация. Протокол TCP обеспечивает работу одновременно нескольких соединений. Каждый прикладной процесс идентифицируется *номером порта*. Заголовок TCP-сегмента содержит номера портов процесса-отправителя и процесса-получателя. При получении сегмента модуль TCP анализирует номер порта получателя и отправляет данные соответствующему прикладному процессу.

Все распространенные сервисы Интернет имеют стандартизованные номера портов. Например, номер порта сервера электронной почты - 25, сервера FTP - 21. Список стандартных номеров портов можно найти в файле /etc/services (Unix).

Совокупность IP-адреса и номера порта является *составным адресом*, который уникально идентифицирует прикладной процесс.

Управление соединениями. Соединение – это совокупность информации о состоянии потока данных, включающая составные адреса, номера посланных, принятых и подтвержденных октетов, размеры окон.

Соединение характеризуется для клиента именем, которое является указателем на структуру TCB (Transmission Control Block), содержащую информацию о соединении.

Открытие соединения клиентом осуществляется вызовом функции OPEN, которой передается составной адрес, с которым требуется установить соединение. Функция возвращает имя соединения. Различают два типа открытия соединения: активное и пассивное. При активном открытии TCP-модуль начинает процедуру установления соединения с указанным адресом, при пассивном - ожидает, что удаленный TCP-модуль начнет процедуру установления соединения с указанного адреса. Указание 0.0.0.0:0 в качестве адреса при пассивном открытии означает, что ожидается соединение с любого адреса. Клиент же применяет процедуру активного открытия; адрес при этом формируется из IP-адреса сервера и стандартного номера порта для данного сервиса. Закрытие соединения клиентом производится с помощью функции CLOSE, которой передается имя соединения.

Процедура установления соединения происходит следующим образом (рисунок 2.19). Предположим, узел A желает установить соединение с узлом B. Первый отправляемый из A в B TCP-сегмент не содержит полезных данных, а

служит для установления соединения. В его заголовке (в поле Flags) установлен бит SYN, означающий запрос связи, и содержится ISN (Initial Sequence Number - начальный номер последовательности) - число, начиная с которого узел А будет нумеровать отправляемые октеты (например, 0). В ответ на получение такого сегмента узел В откликается посылкой TCP-сегмента, в заголовке которого установлен бит ACK, подтверждающий установление соединения для получения данных от узла А. Так как протокол TCP обеспечивает полнодуплексную передачу данных, то узел В в этом же сегменте устанавливает бит SYN, означающий запрос связи для передачи данных от В к А, и передает свой ISN (например, 0). Полезных данных этот сегмент также не содержит. Третий TCP-сегмент в сеансе посылается из А в В в ответ на сегмент, полученный из В. Так как соединение А -> В можно считать установленным (получено подтверждение от В), то узел А включает в свой сегмент полезные данные, нумерация которых начинается с номера ISN(A)+1. Данные нумеруются по количеству отправленных октетов. В заголовке этого же сегмента узел А устанавливает бит ACK, подтверждающий установление связи В -> А, что позволяет системе В включить в свой следующий сегмент полезные данные для А.

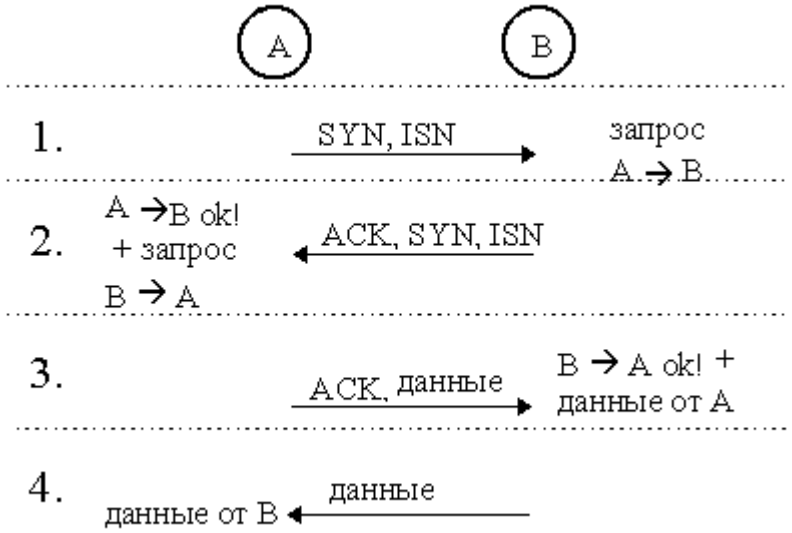


Рисунок 2.19 – Установка TCP-соединения

Сеанс обмена данными заканчивается процедурой разрыва соединения, которая аналогична процедуре установки, с той разницей, что вместо SYN для разрыва используется служебный бит FIN (“данных для отправки больше не имею”), который устанавливается в заголовке последнего сегмента с данными, отправляемого узлом.

Управление потоком. Для ускорения и оптимизации процесса передачи больших объемов данных протокол TCP определяет метод управления потоком, называемый методом скользящего окна, который позволяет отправителю

посылать очередной сегмент, не дожидаясь подтверждения о получении в пункте назначения предшествующего сегмента.

Протокол TCP формирует подтверждения не для каждого конкретного успешно полученного пакета, а для всех данных от начала посылки до некоторого порядкового номера ACK SN (Acknowledge Sequence Number) исключительно. В качестве подтверждения успешного приема, например, первых 2000 байт, высылается ACK SN = 2001: это означает, что все данные в байтовом потоке под номерами от ISN+1=1 до данного ACK SN -1 (2000) успешно получены (рисунок 2.20).

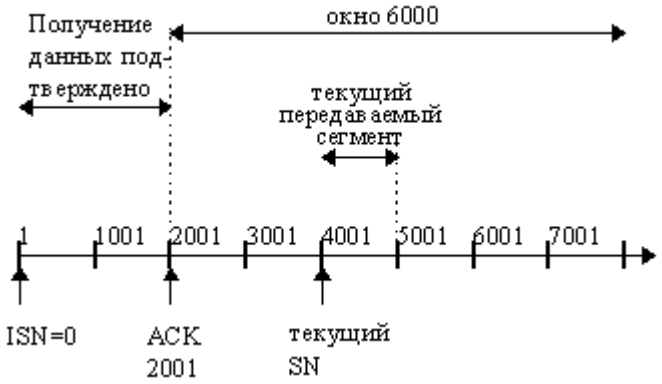


Рисунок 2.20 – Метод скользящего окна

Вместе с посылкой отправителю ACK SN получатель объявляет также “размер окна”, например - 6000. Это значит, что отправитель может посылать данные с порядковыми номерами от текущего ACK SN = 2001 до (ACK SN + размер окна -1) = 8000, не дожидаясь подтверждения со стороны получателя. Допустим, в данный момент отправитель посылает тысячеоктетный сегмент с порядковым номером данных SN=4001. Если не будет получено новое подтверждение (новый ACK SN), отправитель будет посылать данные, пока он остается в пределах объявленного окна, то есть до номера 8001. После этого посылка данных будет прекращена до получения очередного подтверждения и (возможно) нового размера окна. Однако размер окна выбирается таким образом, чтобы подтверждения успевали приходить вовремя и остановки передачи не происходило - для этого и предназначен метод скользящего окна. Размер окна может динамически изменяться получателем.

Для временной остановки посылки данных достаточно объявить нулевое окно. Но даже и в этом случае через определенные промежутки времени будут отправляться сегменты с одним октетом данных. Это делается для того, чтобы отправитель гарантированно узнал о том, что получатель вновь объявил ненулевое окно, поскольку получатель обязан подтвердить получение “пробных” сегментов, а в этих подтверждениях он укажет также и текущий размер своего окна.

Как уже было сказано выше, протокол TCP позволяет вести полнодуплексную передачу. Один и тот же сегмент, высылаемый, например, из В

в А, может содержать в заголовке служебную информацию по подтверждению получения данных от А, а в поле данных - полезные данные для А.

Модуль TCP может оптимизировать максимальный размер сегмента исходя из значений MTU на разных участках маршрута и других характеристик соединения. Модуль TCP может использовать алгоритм "медленного старта", формируя при установлении соединения окно перегрузки, размер которого изначально равен размеру одного сегмента. Это окно показывает, сколько сегментов TCP-модуль, с его собственной точки зрения, может отправить без получения подтверждения. Скользящее же окно, рассмотренное выше, показывает, какой объем неподтвержденных данных модулю разрешено отправить с точки зрения удаленного модуля, получателя его данных. После прихода подтверждения от получателя окно перегрузки увеличивается на 1 сегмент, и отправитель может выслать уже два сегмента, не дожидаясь подтверждения. Такой подход позволяет постепенно увеличивать нагрузку на сеть. Если окно перегрузки становится больше скользящего окна, объявляемого получателем, ограничение на передачу неподтвержденных данных устанавливает уже скользящее окно получателя.

В случае, если никакие данные приложениями не передаются, а соединение открыто, модуль TCP может периодически посылать сегменты-зонды для выяснения того, не отключилась ли другая сторона без уведомления партнера (например, в результате обрыва линии или другим некорректным образом). Такое зондирование проводится примерно каждые два часа неактивности.

Заголовок TCP-сегмента. TCP-сегмент состоит из заголовка и данных. Заголовок сегмента состоит из 32-разрядных слов и имеет переменную длину, зависящую от размера поля Options, но всегда кратную 32 битам (рисунок 2.21). За заголовком непосредственно следуют данные - часть потока данных пользователя, передаваемая в данном сегменте.

Source Port (16 бит), *Destination Port* (16 бит) – номера портов процесса-отправителя и процесса-получателя соответственно.

Sequence Number (SN) (32 бита) – порядковый номер первого октета в поле данных сегмента среди всех октетов потока данных для текущего соединения, то есть если в сегменте пересылаются октеты с 2001-го по 3000-й, то SN=2001. Если в заголовке сегмента установлен бит SYN (фаза установления соединения), то в поле SN записывается начальный номер (ISN), например, 0. Номер первого октета данных, посылаемых после завершения фазы установления соединения, равен ISN+1.

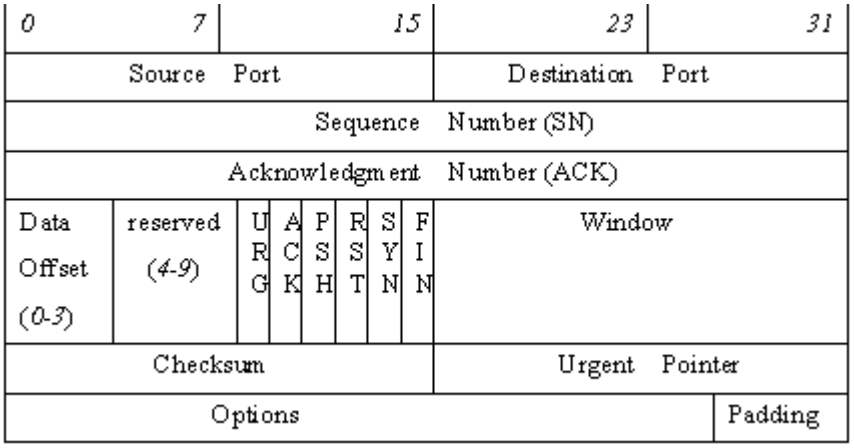


Рисунок 2.21 - Формат заголовка сегмента TCP

Acknowledgment Number (ACK) (32 бита) – если установлен бит ACK, то это поле содержит порядковый номер октета, который отправитель данного сегмента желает получить. Это означает, что все предыдущие октеты (с номерами от ISN+1 до ACK-1 включительно) были успешно получены.

Data Offset (4 бита) - длина TCP-заголовка в 32-битных словах.

Reserved (6 бит) - зарезервировано; заполняется нулями.

Control Bits (6 бит) - управляющие биты; активным является положение “бит установлен”:

- *URG* – поле срочного указателя (Urgent Pointer) задействовано;
- *ACK* – поле номера подтверждения (Acknowledgment Number) задействовано;
- *PSH* – осуществить “проталкивание” - если модуль TCP получает сегмент с установленным флагом PSH, то он немедленно передает все данные из буфера приема процессу-получателю для обработки, даже если буфер не был заполнен;
- *RST* – перезагрузка текущего соединения;
- *SYN* – запрос на установление соединения;
- *FIN* – нет больше данных для передачи.

Window (16 бит) - размер окна в октетах.

Checksum (16 бит) – контрольная сумма, представляет собой 16 бит, дополняющие биты в сумме всех 16-битовых слов сегмента (само поле контрольной суммы перед вычислением обнуляется). Контрольная сумма, кроме заголовка сегмента и поля данных, учитывает 96 бит псевдозаголовка, который для внутреннего употребления ставится перед TCP-заголовком. Этот псевдозаголовок содержит IP-адрес отправителя (4 октета), IP-адрес получателя (4 октета), нулевой октет, 8-битное поле "Протокол", аналогичное полю в IP-заголовке, и 16 бит длины TCP сегмента, измеренной в октетах. Такой подход обеспечивает защиту протокола TCP от ошибшихся в маршруте сегментов. Информация для псевдозаголовка передается через интерфейс "Протокол TCP/межсетевой уровень" в качестве аргументов или результатов запросов от протокола TCP к протоколу IP.

Urgent Pointer (16 бит) – используется для указания длины срочных данных, которые размещаются в начале поля данных сегмента. Указывает смещение октета, следующего за срочными данными, относительно первого октета в сегменте. Например, в сегменте передаются октеты с 2001-го по 3000-й, при этом первые 100 октетов являются срочными данными, тогда Urgent Pointer = 100. Протокол TCP не определяет, как именно должны обрабатываться срочные денные, но предполагает, что прикладной процесс будет предпринимать усилия для их быстрой обработки. Поле Urgent Pointer задействовано, если установлен флаг URG.

Options – поле переменной длины; может отсутствовать или содержать одну опцию или список опций, реализующих дополнительные услуги протокола TCP.

Padding – выравнивание заголовка по границе 32-битного слова, если список опций занимает нецелое число 32-битных слов. Поле Padding заполняется нулями.

Промежуточные состояния соединения. TCP-соединение во время функционирования проходит через ряд промежуточных состояний. Это состояния LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, а также фиктивное состояние CLOSED. (Состояние CLOSED является фиктивным, поскольку оно представляет отсутствие соединения.) Переход из одного состояния в другое происходит в ответ на события, как то: запросы клиента, приход сегментов, истечение контрольного времени.

Определены следующие запросы процесса-клиента модулю TCP (с каждым запросом, кроме OPEN, передается имя соединения):

- ACTIVE-OPEN – активное открытие соединения;
- PASSIVE-OPEN – пассивное открытие соединения;
- SEND – отправка данных (передается указатель на буфер данных, размер буфера, значения флагов URG и PSH);
- RECEIVE – получение данных (передается указатель на буфер данных, размер буфера; возвращается счетчик полученных октетов, значения флагов URG и PSH);
- STATUS – запрос состояния соединения;
- CLOSE – закрытие соединения (производится досылка всех неотправленных данных и обмен сегментами с битом FIN);
- ABORT – ликвидация соединения (уничтожаются блок TCB и все неотправленные данные, посылается сегмент с битом RST).

Деятельность программы протокола TCP можно рассматривать как реагирование на события в зависимости от состояния соединения. Ниже описаны состояния соединения и приведены диаграммы переходов. Под термином "процесс" здесь понимается процесс TCP-модуля, работающий с данным соединением на локальном узле; термин "чужой" относится к процессу, работающему с данным TCP-соединением на удаленном узле.

LISTEN – процесс пассивно ждет запроса со стороны чужих сокетов.

SYN-SENT – процесс отправил свой SYN и ждет чужого SYN.

SYN-RECEIVED – процесс получил чужой SYN, отправил (раньше или только что) свой SYN и ждет ACK на свой SYN.

ESTABLISHED – процесс отправил ACK на чужой SYN, получил ACK на свой SYN; соединение установлено.

FIN-WAIT-1 – процесс первый отправил свой FIN и ждет реакцию той стороны; при этом он, возможно, продолжает получать данные.

FIN-WAIT-2 – процесс получил ACK на свой ранее отправленный FIN, но не получил чужой FIN; ждет чужой FIN; при этом, возможно, продолжает получать данные.

CLOSE-WAIT – процесс, не отправив свой FIN (возможно, не собираясь прекращать соединение), получает чужой FIN; он отправляет ACK на чужой FIN, но при этом, возможно, продолжает отправлять данные.

LAST-ACK – процесс отправил свой FIN, но ранее он уже получил FIN с той стороны и отправил на него ACK; поэтому процесс ожидает чужой ACK на свой FIN для окончательного закрытия соединения.

CLOSING – процесс ранее отправил свой FIN и еще не получил не него подтверждение, но получил чужой FIN (и отправил на него ACK); ждет ACK на свой FIN.

TIME-WAIT – процесс ранее отправил свой FIN и получил на него подтверждение, получил чужой FIN и только что отправил на него ACK; теперь процесс ждет некоторое время (два времени жизни сегмента, обычно 4 минуты) для гарантии того, что та сторона получит его ACK на свой FIN, после чего соединение будет окончательно закрыто.

CLOSED – соединение отсутствует.

На рисунке 2.21 и рисунке 2.22 приведены фазы установления и закрытия соединения.

На рисунке 2.21 и рисунке 2.22 состояния соединения заключены в прямоугольники, переходы между ними показаны стрелками, с каждой стрелкой соотносится овал, поясняющий причину перехода. В овале над горизонтальной чертой указывается событие, вызвавшее переход (курсивом обозначено поступление запросов от локального процесса-клиента); под горизонтальной чертой - действие, сопутствующее переходу.

Проблемы возникновения некорректных ситуаций, например, наполовину открытое соединение, получение заблудившихся в сети старых SYN-сегментов, неожиданный крах программ и т.п., решаются путем детектирования ошибки (несоответствие или бессмысленные значения ACK или SN), после чего посылается сигнал RST (сегмент с установленным битом RST) и соединение ликвидируется.

Адаптивные свойства протокола TCP. Выбор времени ожидания (тайм-аута) очередной квитанции является важной задачей, результат решения которой влияет на производительность протокола TCP. Тайм-аут не должен быть слишком коротким, чтобы по возможности исключить избыточные повторные передачи, которые снижают полезную пропускную способность системы.

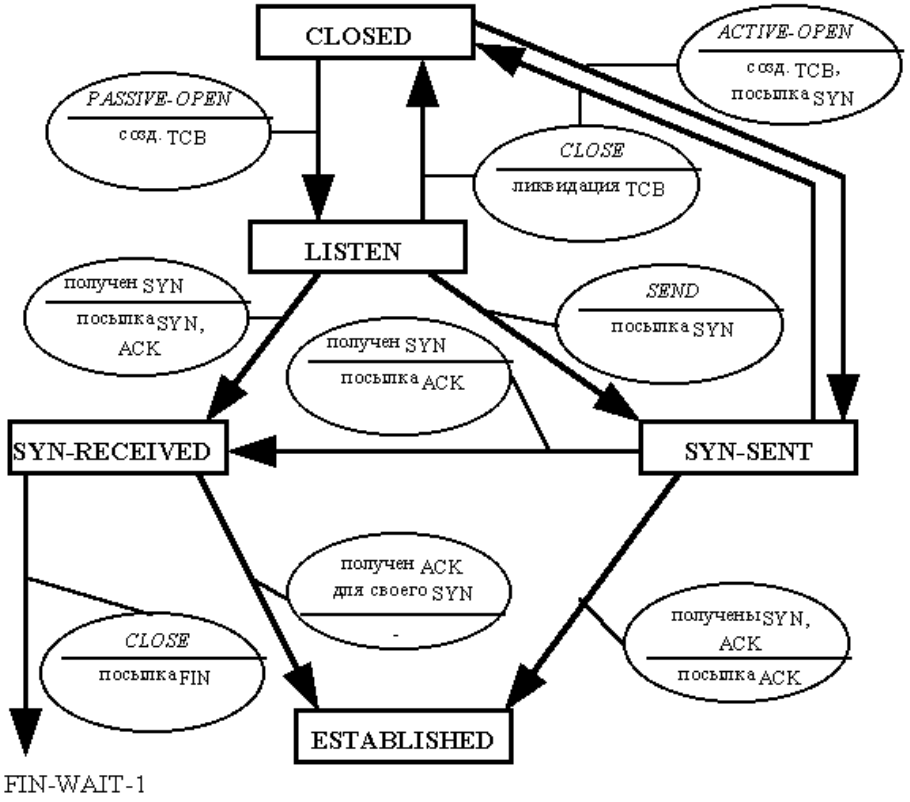


Рисунок 2.21 – Фаза установления соединения

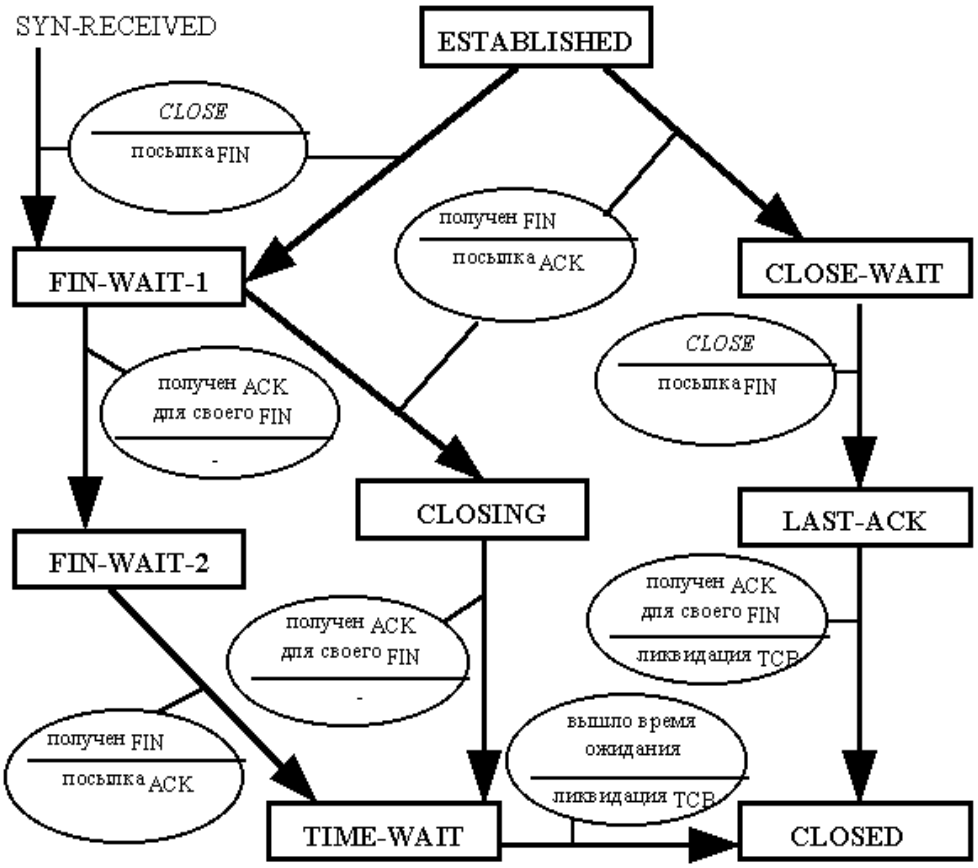


Рисунок 2.22 – Фаза закрытия соединения

Но он не должен быть и слишком большим, чтобы избежать длительных простоев, связанных с ожиданием несуществующей или «заблудившейся» квитанции.

При выборе величины тайм-аута должны учитываться скорость и надежность физических линий связи, их протяженность и многие другие подобные факторы. В протоколе TCP тайм-аут определяется с помощью достаточно сложного адаптивного алгоритма, идея которого состоит в следующем. При каждой передаче засекается время от момента отправки сегмента до прихода квитанции о его приеме (время оборота). Получаемые значения времени оборота усредняются с весовыми коэффициентами, возрастающими от предыдущего замера к последующему. Это делается с тем, чтобы усилить влияние последних замеров. В качестве тайм-аута выбирается среднее время оборота, умноженное на некоторый коэффициент. Практика показывает, что значение этого коэффициента должно превышать 2. В сетях с большим разбросом времени оборота при выборе тайм-аута учитывается и дисперсия этой величины.

Поскольку каждый байт пронумерован, то каждый из них может быть опознан. Приемлемый механизм опознавания является накопительным, поэтому опознавание номера X означает, что все байты с предыдущими номерами уже получены. Этот механизм позволяет регистрировать появление дубликатов в условиях повторной передачи. Нумерация байтов в пределах сегмента осуществляется так, чтобы первый байт данных сразу вслед за заголовком имел наименьший номер, а следующие за ним байты имели номера по возрастающей.

Окно, посылаемое с каждым сегментом, определяет диапазон номеров очереди, которые отправитель окна (он же получатель данных) готов принять в настоящее время. Предполагается, что такой механизм связан с наличием в данный момент места в буфере данных.

Варьируя величину окна, можно влиять на загрузку сети. Чем больше окно, тем большую порцию неподтвержденных данных можно послать в сеть. Но если пришло большее количество данных, чем может быть принято программой TCP, данные будут отброшены. Это приведет к излишним пересылкам информации и ненужному увеличению нагрузки на сеть и программу TCP.

С другой стороны, указание окна малого размера может ограничить передачу данных скоростью, которая определяется временем путешествия по сети каждого посылаемого сегмента. Чтобы избежать применения малых окон, получателю данных предлагается откладывать изменение окна до тех пор, пока свободное место не составит 20-40 % от максимально возможного объема памяти для этого соединения. Но и отправителю не стоит спешить с посылкой данных, пока окно не станет достаточно большим. Учитывая эти соображения, разработчики протокола TCP предложили схему, согласно которой при установлении

соединения заявляется большое окно, но впоследствии его размер существенно уменьшается.

Если сеть не справляется с нагрузкой, то возникают очереди в промежуточных узлах – маршрутизаторах и в конечных узлах-компьютерах.

При переполнении приемного буфера конечного узла «перегруженный» протокол TCP, отправляя квитанцию, помещает в нее новый, уменьшенный размер окна. Если он совсем отказывается от приема, то в квитанции указывается окно нулевого размера. Однако даже после этого приложение может послать сообщение на отказавшийся от приема порт. Для этого сообщение должно сопровождаться пометкой «срочно». В такой ситуации порт обязан принять сегмент, даже если для этого придется вытеснить из буфера уже находящиеся там данные. После приема квитанции с нулевым значением окна протокол-отправитель время от времени делает контрольные попытки продолжить обмен данными. Если протокол-приемник уже готов принимать информацию, то в ответ на контрольный запрос он посылает квитанцию с указанием ненулевого размера окна.

Другим проявлением перегрузки сети является переполнение буферов в маршрутизаторах. В таких случаях они могут централизованно изменить размер окна, посылая управляющие сообщения некоторым конечным узлам, что позволяет им дифференцированно управлять интенсивностью потока данных в разных частях сети.

10.1.2 Протокол UDP

UDP (User Datagram Protocol, протокол пользовательских дейтаграмм) фактически не выполняет каких-либо особых функций дополнительно к функциям межсетевого уровня (протокола IP). Протокол UDP используется либо при пересылке коротких сообщений, когда накладные расходы на установление сеанса и проверку успешной доставки данных оказываются выше расходов на повторную (в случае неудачи) пересылку сообщения, либо в том случае, когда сама организация процесса-приложения обеспечивает установление соединения и проверку доставки пакетов (например, NFS).

Пользовательские данные, поступившие от прикладного уровня, предваряются UDP-заголовком, и сформированный таким образом UDP-пакет отправляется на межсетевой уровень. UDP-заголовок состоит из двух 32-битных слов (рисунок 2.23).

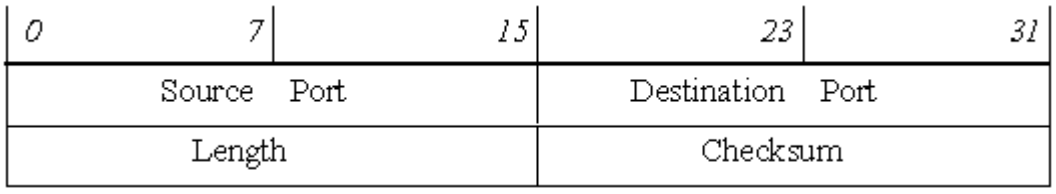


Рисунок 2.23 – Заголовок UDP

Source Port – номер порта процесса-отправителя.

Destination Port – номер порта процесса-получателя.

Length – длина UDP-пакета вместе с заголовком в октетах.

Checksum – контрольная сумма. Контрольная сумма вычисляется таким же образом, как и в TCP-заголовке; если UDP-пакет имеет нечетную длину, то при вычислении контрольной суммы к нему добавляется нулевой октет.

После заголовка непосредственно следуют пользовательские данные, переданные модулю UDP прикладным уровнем за один вызов. Протокол UDP рассматривает эти данные как целостное сообщение; он никогда не разбивает сообщение для передачи в нескольких пакетах и не объединяет несколько сообщений для пересылки в одном пакете. Если прикладной процесс N раз вызвал модуль UDP для отправки данных (т.е. запросил отправку N сообщений), то модулем UDP будет сформировано и отправлено N пакетов, и процесс-получатель будет должен N раз вызвать свой модуль UDP для получения всех сообщений.

При получении пакета от межсетевого уровня модуль UDP проверяет контрольную сумму и передает содержимое сообщения прикладному процессу, чей номер порта указан в поле “Destination Port”.

Если проверка контрольной суммы выявила ошибку или если процесса, подключенного к требуемому порту, не существует, пакет игнорируется. Если пакеты поступают быстрее, чем модуль UDP успевает их обрабатывать, то поступающие пакеты также игнорируются. Протокол UDP не имеет никаких средств подтверждения безошибочного приема данных или сообщения об ошибке, не обеспечивает приход сообщений в порядке отправки, не производит предварительного установления сеанса связи между прикладными процессами, поэтому он является *ненадежным* протоколом *без установления соединения*. Если приложение нуждается в подобного рода услугах, оно должно использовать на транспортном уровне протокол TCP.

Максимальная длина UDP-сообщения равна максимальной длине IP-дейтаграммы (65535 октетов) за вычетом минимального IP-заголовка (20) и UDP-заголовка (8), т.е. 65507 октетов. На практике обычно используются сообщения длиной 8192 октета.

Примеры прикладных процессов, использующих протокол UDP: NFS (Network File System - сетевая файловая система), TFTP (Trivial File Transfer Protocol - простой протокол передачи файлов), SNMP (Simple Network Management Protocol - простой протокол управления сетью), DNS (Domain Name Service - доменная служба имен).

11 Базовые технологии локальных сетей

11.1 Протоколы и стандарты локальных сетей

Общая характеристика протоколов локальных сетей. При организации взаимодействия узлов в локальных сетях основная роль отводится протоколу канального уровня. Однако для того, чтобы канальный уровень мог справиться с этой задачей, структура локальных сетей должна быть вполне определенной, так, например, наиболее популярный протокол канального уровня — Ethernet — рассчитан на параллельное подключение всех узлов сети к общей для них шине — отрезку коаксиального кабеля или иерархической древовидной структуре сегментов, образованных повторителями. Протокол Token Ring также рассчитан на вполне определенную конфигурацию — соединение компьютеров в виде логического кольца.

Подобный подход, заключающийся в использовании простых структур кабельных соединений между компьютерами локальной сети, соответствовал основной цели, которую ставили перед собой разработчики первых локальных сетей во второй половине 70-х годов. Эта цель заключалась в нахождении простого и дешевого решения для объединения в вычислительную сеть нескольких десятков компьютеров, находящихся в пределах одного здания.

Для упрощения и, соответственно, удешевления аппаратных и программных решений разработчики первых локальных сетей остановились на совместном использовании кабелей всеми компьютерами сети в режиме разделения времени, то есть режиме TDM. Наиболее явным образом режим совместного использования кабеля проявляется в классических сетях Ethernet, где коаксиальный кабель физически представляет собой неделимый отрезок кабеля, общий для всех узлов сети. Но и в сетях Token Ring и FDDI, где каждая соседняя пара компьютеров соединена, казалось бы, своими индивидуальными отрезками кабеля с концентратором, эти отрезки не могут использоваться компьютерами, которые непосредственно к ним подключены, в произвольный момент времени. Эти отрезки образуют логическое кольцо, доступ к которому как к единому целому может быть получен только по вполне определенному алгоритму, в котором участвуют все компьютеры сети. Использование кольца как общего разделяемого ресурса упрощает алгоритмы передачи по нему кадров, так как в каждый конкретный момент времени кольцо занято только одним компьютером.

Использование разделяемых сред (shared media) позволяет упростить логику работы сети. Например, отпадает необходимость контроля переполнения узлов сети кадрами от многих станций, решивших одновременно обменяться информацией. В глобальных сетях, где отрезки кабелей, соединяющих отдельные узлы, не рассматриваются как общий ресурс, такая необходимость возникает, и

для решения этой проблемы в протоколы обмена информацией вводятся весьма сложные процедуры управления потоком кадров, предотвращающие переполнение каналов связи и узлов сети.

Использование в локальных сетях очень простых конфигураций (общая шина и кольцо) наряду с положительными имело и отрицательные последствия, из которых наиболее неприятными были ограничения по производительности и надежности. Наличие только одного пути передачи информации, разделяемого всеми узлами сети, в принципе ограничивало пропускную способность сети пропускной способностью этого пути (которая делилась в среднем на число компьютеров сети), а надежность сети — надежностью этого пути. Поэтому по мере повышения популярности локальных сетей и расширения их сфер применения все больше стали применяться специальные коммуникационные устройства — мосты и маршрутизаторы, — которые в значительной мере снимали ограничения единственной разделяемой среды передачи данных. Базовые конфигурации в форме общей шины и кольца превратились в элементарные структуры локальных сетей, которые можно теперь соединять друг с другом более сложным образом, образуя параллельные основные или резервные пути между узлами.

Тем не менее внутри базовых структур по-прежнему работают все те же протоколы разделяемых единственных сред передачи данных, которые были разработаны более 15 лет назад. Это связано с тем, что хорошие скоростные и надежностные характеристики кабелей локальных сетей удовлетворяли в течение всех этих лет пользователей небольших компьютерных сетей, которые могли построить сеть без больших затрат только с помощью сетевых адаптеров и кабеля. К тому же колоссальная инсталляционная база оборудования и программного обеспечения для технологий Ethernet и Token Ring способствовала тому, что сложился следующий подход: в пределах небольших сегментов используются старые протоколы в их неизменном виде, а объединение таких сегментов в общую сеть происходит с помощью дополнительного и достаточно сложного оборудования. В последние несколько лет наметилось движение к отказу от разделяемых сред передачи данных в локальных сетях и переходу к применению активных коммутаторов, к которым конечные узлы присоединяются индивидуальными линиями связи. В чистом виде такой подход предлагается в технологии ATM (Asynchronous Transfer Mode), а в технологиях, носящих традиционные названия с приставкой switched (коммутируемый): switched Ethernet, switched Token Ring, switched FDDI, обычно используется смешанный подход, сочетающий разделяемые и индивидуальные среды передачи данных. Чаще всего конечные узлы соединяются в небольшие разделяемые сегменты с помощью повторителей, а сегменты соединяются друг с другом с помощью индивидуальных коммутируемых связей.

Существует и достаточно заметная тенденция к использованию в традиционных технологиях так называемой микросегментации, когда даже конечные узлы сразу соединяются с коммутатором индивидуальными каналами. Такие сети получаются дороже разделяемых или смешанных, но производительность их выше.

При использовании коммутаторов у традиционных технологий появился новый режим работы — *полнодуплексный (full-duplex).* В разделяемом сегменте станции всегда работают в *полудуплексном режиме (half-duplex),* так как в каждый момент времени сетевой адаптер станции либо передает свои данные, либо принимает чужие, но никогда не делает это одновременно. Это справедливо для всех технологий локальных сетей, так как разделяемые среды поддерживаются не только классическими технологиями локальных сетей Ethernet, Token Ring, FDDI, но и всеми новыми — Fast Ethernet, lOOVG-AnyLAN, Gigabit Ethernet.

В полнодуплексном режиме сетевой адаптер может одновременно передавать свои данные в сеть и принимать из сети чужие данные. Такой режим несложно обеспечивается при прямом соединение с мостом/коммутатором или маршрутизатором, так как вход и выход каждого порта такого устройства работают независимо друг от друга, каждый со своим буфером кадров.

Сегодня каждая технология локальных сетей приспособлена для работы как в полудуплексном, так и полнодуплексном режимах. В этих режимах ограничения, накладываемые на общую длину сети, существенно отличаются, так что одна и та же технология может позволять строить весьма различные сети в зависимости от выбранного режима работы (который зависит от того, какие устройства используются для соединения узлов — повторители или коммутаторы). Например, технология Fast Ethernet позволяет для полудуплексного режима строить сети диаметром не более 200 метров, а для полнодуплексного режима ограничений на диаметр сети не существует. Поэтому при сравнении различных технологий необходимо обязательно принимать во внимание возможность их работы в двух режимах. В данной главе изучается в основном полудуплексный режим работы протоколов, а полнодуплексный режим рассматривается в следующей главе, совместно с изучением коммутаторов.

Несмотря на появление новых технологий, классические протоколы локальных сетей Ethernet и Token Ring по прогнозам специалистов будут повсеместно использоваться еще по крайней мере лет 5-10, в связи с чем знание их деталей необходимо для успешного применения современной коммуникационной аппаратуры. Кроме того, некоторые современные высокопроизводительные технологии, такие как Fast Ethernet, Gigabit Ethernet, в значительной степени сохраняют преемственность со своими предшественниками. Это еще раз подтверждает важность изучения классических протоколов локальных сетей, естественно, наряду с изучением новых технологий.

11.2 Структура стандартов IEEE 802.x

В 1980 году в институте IEEE был организован комитет 802 по стандартизации локальных сетей, в результате работы которого было принято семейство стандартов IEEE 802.x, которые содержат рекомендации по проектированию нижних уровней локальных сетей. Позже результаты работы этого комитета легли в основу комплекса международных стандартов ISO 8802-1...5. Эти стандарты были созданы на основе очень распространенных фирменных стандартов сетей Ethernet, ArcNet и Token Ring.

Помимо IEEE в работе по стандартизации протоколов локальных сетей принимали участие и другие организации. Так, для сетей, работающих на оптоволокне, американским институтом по стандартизации ANSI был разработан стандарт FDDI, обеспечивающий скорость передачи данных 100 Мб/с. Работы по стандартизации протоколов ведутся также ассоциацией ECMA, которой приняты стандарты ECMA-80, 81, 82 для локальной сети типа Ethernet и впоследствии стандарты ECMA-89,90 по методу передачи маркера.

Стандарты семейства IEEE 802.x охватывают только два нижних уровня семиуровневой модели OSI — физический и канальный. Это связано с тем, что именно эти уровни в наибольшей степени отражают специфику локальных сетей. Старшие же уровни, начиная с сетевого, в значительной степени имеют общие черты как для локальных, так и для глобальных сетей.

Специфика локальных сетей также нашла свое отражение в разделении канального уровня на два подуровня, которые часто называют также уровнями. Канальный уровень (Data Link Layer) делится в локальных сетях на два подуровня:

- логической передачи данных (Logical Link Control, LLC);
- управления доступом к среде (Media Access Control, MAC).

Уровень MAC появился из-за существования в локальных сетях разделяемой среды передачи данных. Именно этот уровень обеспечивает корректное совместное использование общей среды, предоставляя ее в соответствии с определенным алгоритмом в распоряжение той или иной станции сети. После того как доступ к среде получен, ею может пользоваться более высокий уровень — уровень LLC, организующий передачу логических единиц данных, кадров информации, с различным уровнем качества транспортных услуг. В современных локальных сетях получили распространение несколько протоколов уровня MAC, реализующих различные алгоритмы доступа к разделяемой среде. Эти протоколы полностью определяют специфику таких технологий, как Ethernet, Fast Ethernet, Gigabit Ethernet, Token Ring, FDDI, lOOVG-AnyLAN.

Уровень LLC отвечает за передачу кадров данных между узлами с различной степенью надежности, а также реализует функции интерфейса с прилегающим к нему сетевым уровнем. Именно через уровень LLC сетевой протокол запрашивает

у канального уровня нужную ему транспортную операцию с нужным качеством. На уровне LLC существует несколько режимов работы, отличающихся наличием или отсутствием на этом уровне процедур восстановления кадров в случае их потери или искажения, то есть отличающихся качеством транспортных услуг этого уровня.

Протоколы уровней MAC и LLC взаимно независимы — каждый протокол уровня MAC может применяться с любым протоколом уровня LLC, и наоборот. Стандарты IEEE 802 имеют достаточно четкую структуру, приведенную на рисунке 2.24.

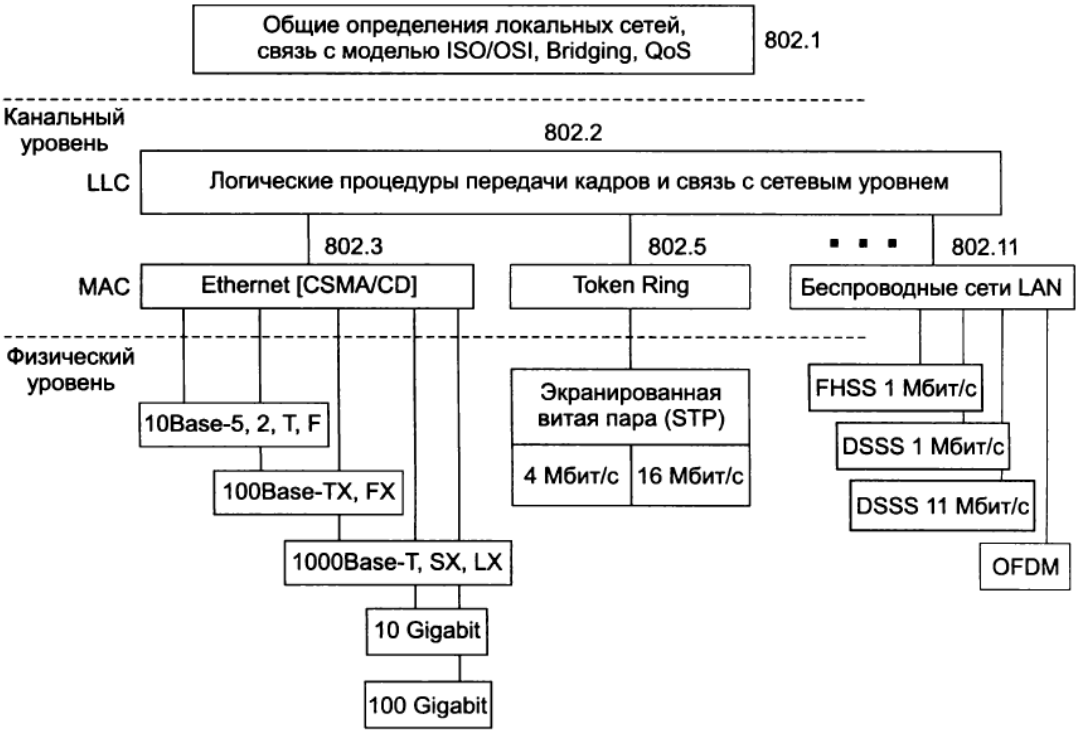


Рисунок 2.24 – Структура стандартов IEEE 802.x

Эта структура появилась в результате большой работы, проведенной комитетом 802 по выделению в разных фирменных технологиях общих подходов и общих функций, а также согласованию стилей их описания. В результате канальный уровень был разделен на два упомянутых подуровня. Описание каждой технологии разделено на две части: описание уровня MAC и описание физического уровня. Как видно из рисунка, практически у каждой технологии единственному протоколу уровня MAC соответствует несколько вариантов протоколов физического уровня (на рисунке в целях экономии места приведены только технологии Ethernet и Token Ring, но все сказанное справедливо также и для остальных технологий, таких как ArcNet, FDDI, lOOVG-AnyLAN).

Над канальным уровнем всех технологий изображен общий для них протокол LLC, поддерживающий несколько режимов работы, но независимый от выбора

конкретной технологии. Стандарт LLC курирует подкомитет 802.2. Даже технологии, стандартизованные не в рамках комитета 802, ориентируются на использование протокола LLC, определенного стандартом 802.2, например, протокол FDDI, стандартизованный ANSI.

Особняком стоят стандарты, разрабатываемые подкомитетом 802.1.'Эти стандарты носят общий для всех технологий характер. В подкомитете 802.1 были разработаны общие определения локальных сетей и их свойств, определена связь трех уровней модели IEEE 802 с моделью OSI. Но наиболее практически важными являются стандарты 802.1, которые описывают взаимодействие между собой различных технологий, а также стандарты по построению более сложных сетей на основе базовых топологий. Эта группа стандартов носит общее название стандартов межсетевого взаимодействия (internetworking). Сюда входят такие важные стандарты, как стандарт 802.1D, описывающий логику работы моста/коммутатора, стандарт 802.1H, определяющий работу транслирующего моста, который может без маршрутизатора объединять сети Ethernet и FDDI, Ethernet и Token Ring и т. п. Сегодня набор стандартов, разработанных подкомитетом 802.1, продолжает расти. Например, недавно он пополнился важным стандартом 802.1Q, определяющим способ построения виртуальных локальных сетей VLAN в сетях на основе коммутаторов.

Стандарты 802.3, 802.4, 802.5 и 802.12 описывают технологии локальных сетей, которые появились в результате улучшений фирменных технологий, легших в их основу. Так, основу стандарта 802.3 составила технология Ethernet, разработанная компаниями Digital, Intel и Xerox (или Ethernet DIX), стандарт 802.4 появился как обобщение технологии ArcNet компании Datapoint Corporation, а стандарт 802.5 в основном соответствует технологии Token Ring компании IBM.

Исходные фирменные технологии и их модифицированные варианты — стандарты 802.x в ряде случаев долгие годы существовали параллельно. Например, технология ArcNet так до конца не была приведена в соответствие со стандартом 802.4 (теперь это делать поздно, так как где-то примерно с 1993 года производство оборудования ArcNet было свернуто). Расхождения между технологией Token Ring и стандартом 802.5 тоже периодически возникают, так как компания IBM регулярно вносит усовершенствования в свою технологию и комитет 802.5 отражает эти усовершенствования в стандарте с некоторым запозданием. Исключение составляет технология Ethernet. Последний фирменный стандарт Ethernet DIX был принят в 1980 году, и с тех пор никто больше не предпринимал попыток фирменного развития Ethernet. Все новшества в семействе технологий Ethernet вносятся только в результате принятия открытых стандартов комитетом 802.3.

Более поздние стандарты изначально разрабатывались не одной компанией, а группой заинтересованных компаний, а потом передавались в соответствующий

подкомитет IEEE 802 для утверждения. Так произошло с технологиями Fast Ethernet, lOOVG-AnyLAN, Gigabit Ethernet. Группа заинтересованных компаний образовывала сначала небольшое объединение, а затем по мере развития работ к нему присоединялись другие компании, так что процесс принятия стандарта носил открытый характер.

Сегодня комитет 802 включает следующий ряд подкомитетов, в который входят как уже упомянутые, так и некоторые другие:

- 802.1 — Internetworking — объединение сетей;
- 802.2 — Logical Link Control, LLC — управление логической передачей данных;
- 802.3 — Ethernet с методом доступа CSMA/CD;
- 802.4 — Token Bus LAN — локальные сети с методом доступа Token Bus;
- 802.5 — Token Ring LAN — локальные сети с методом доступа Token Ring;
- 802.6 — Metropolitan Area Network, MAN — сети мегаполисов;
- 802.7 — Broadband Technical Advisory Group — техническая консультационная группа по широкополосной передаче; • 802.8 — Fiber Optic Technical Advisory Group — техническая консультационная группа по волоконно-оптическим сетям;
- 802.9 — Integrated Voice and data Networks — интегрированные сети передачи голоса и данных;
- 802.10 — Network Security — сетевая безопасность;
- 802.11 — Wireless Networks — беспроводные сети;
- 802.12 — Demand Priority Access LAN, lOOVG-AnyLAN — локальные сети с методом доступа по требованию с приоритетами.

Однако, как видно из рис. 11.3, помимо индивидуальных для каждой технологии уровней существует общий уровень, который был стандартизован рабочей группой 802.2.

Появление этого уровня связано с тем, что комитет 802 разделил функции канального уровня модели OSI на два уровня:

- управления логическим каналом (Logical Link Control, LLC);
- управления доступом к среде (Media Access Control, MAC).

Основными функциями уровня MAC являются:

- обеспечение доступа к разделяемой среде;
- передача кадров между конечными узлами посредством функций и устройств физического уровня.

Если уровень MAC специфичен для каждой технологии и отражает различия в методах доступа к разделяемой среде, то уровень LLC представляет собой обобщение функций разных технологий по обеспечению передачи кадра с различными требованиями к надежности.

Логика образования общего для всех технологий уровня LLC заключается в следующем: после того как узел сети получил доступ к среде в соответствии с алгоритмом, специфическим для конкретной технологии, дальнейшие действия узла или узлов по обеспечению надежной передачи кадров не зависят от этой технологии.

Так как в зависимости от требований приложения может понадобиться разная степень надежности передачи, то рабочая группа 802.2 определила три типа услуг:

Услуга LLC1 — это услуга без установления соединения и без подтверждения получения данных.

Услуга LLC2 дает пользователю возможность установить логическое соединение перед началом передачи любого блока данных, и если это требуется, выполнить процедуры восстановления после ошибок и упорядочивание потока блоков в рамках установленного соединения.

Услуга LLC3 — это услуга без установления соединения, но с подтверждением получения данных.

Какой из трех режимов работы уровня LLC будет использован, зависит от требований протокола верхнего уровня.

Нужно сказать, что на практике идея обобщения функций обеспечения надежной передачи кадров в общем уровне LLC не оправдала себя. Технология Ethernet в версии DIX изначально функционировала в наиболее простом дейтаграммном режиме — в результате оборудование Ethernet и после опубликования стандарта IEEE 802.2 продолжало поддерживать только этот режим работы, который формально является режимом LLC1. В то же время оборудование сетей Token Ring, которое изначально поддерживало режимы LLC2 и LLC3, также продолжало поддерживать эти режимы и никогда не поддерживало режим LLC1. В настоящее время задача обеспечения надежной передачи данных в наибольшей мере возлагается на протокол TCP, в котором реализован механизм контроля потока на основе концепции скользящего окна.

11.3 Технология Ethernet (802.3)

Когда говорят Ethernet, то под этим обычно понимают любой из вариантов этой технологии в более узком смысле Ethernet — это сетевой стандарт, основанный на экспериментальной сети Ethernet Network, которую фирма Xerox разработала и реализовала в 1975 году. Метод доступа был опробован еще раньше: во второй половине 60-х годов в радиосети Гавайского университета использовались различные варианты случайного доступа к общей радиосреде, получившие общее название Aloha. В 1980 году фирмы DEC, Intel и Xerox совместно разработали и опубликовали стандарт Ethernet версии II для сети, построенной на основе коаксиального кабеля, который стал последней версией

фирменного стандарта Ethernet. Поэтому фирменную версию стандарта Ethernet называют стандартом Ethernet DIX или Ethernet II.

На основе стандарта Ethernet DIX был разработан стандарт IEEE 802.3, который во многом совпадает со своим предшественником, но некоторые различия все же имеются. В то время как в стандарте IEEE 802.3 различаются уровни MAC и LLC, в оригинальном Ethernet оба эти уровня объединены в единый канальный уровень. В Ethernet DIX определяется протокол тестирования конфигурации (Ethernet Configuration Test Protocol), который отсутствует в IEEE 802.3. Несколько отличается и формат кадра, хотя минимальные и максимальные размеры кадров в этих стандартах совпадают. Часто для того, чтобы отличить Ethernet, определенный стандартом IEEE, и фирменный Ethernet DIX, первый называют технологией 802.3, а за фирменным оставляют название Ethernet без дополнительных обозначений.

В зависимости от типа физической среды стандарт IEEE 802.3 имеет различные базовые модификации – 10Base-5, 10Base-2, 10Base-T, 10Base-FL, 10Base-FB.

В 1995 году был принят стандарт Fast Ethernet, который во многом не является самостоятельным стандартом, о чем говорит и тот факт, что его описание просто является дополнительным разделом к основному стандарту 802.3 — разделом 802.3и. Аналогично, принятый в 1998 году стандарт Gigabit Ethernet описан в разделе 802.3z основного документа.

Для передачи двоичной информации по кабелю для всех вариантов физического уровня технологии Ethernet, обеспечивающих пропускную способность 10 Мбит/с, используется манчестерский код.

Все виды стандартов Ethernet (в том числе Fast Ethernet и Gigabit Ethernet) используют один и тот же метод разделения среды передачи данных — метод CSMA/CD.

11.3.1 MAC-адреса

На уровне MAC, который обеспечивает доступ к среде и передачу кадра, для идентификации сетевых интерфейсов узлов сети используются регламентированные стандартом IEEE 802.3 уникальные 6-байтовые адреса, называемые MAC-адресами. Обычно MAC- адрес записывают в виде шести пар шестнадцатеричных цифр, разделенных дефисами или двоеточиями, например 11-A0-17-3D-BC-01. Каждый сетевой адаптер имеет по крайней мере один MAC-адрес. Помимо отдельных интерфейсов MAC-адрес может определять группу интерфейсов или даже все интерфейсы сети. Первый (младший) бит старшего байта адреса назначения - это признак того, является адрес индивидуальным или групповым. Если он равен 0, то адрес является индивидуальным, то есть идентифицирует один сетевой интерфейс, а если 1, то групповым. Групповой

адрес связан только с интерфейсами, сконфигурированными (вручную или автоматически по запросу вышележащего уровня) как члены группы, номер которой указан в групповом адресе. Если сетевой интерфейс включен в группу, то наряду с уникальным MAC-адресом с ним ассоциируется еще один адрес - групповой. В частном случае, если групповой адрес состоит из всех единиц, то есть имеет шестнадцатеричное представление OxFFFFFFFFFFF, он идентифицирует все узлы сети и называется широковещательным.

Второй бит старшего байта адреса определяет способ назначения адреса – централизованный или локальный. Если этот бит равен 0 (что бывает почти всегда в стандартной аппаратуре Ethernet), это говорит о том, что адрес назначен централизованно по правилам IEEE 802.

Комитет IEEE распределяет между производителями оборудования так называемые организационно уникальные идентификаторы (Organizationally Unique Identifier, OUI). Каждый производитель помещает выделенный ему идентификатор в три старших байта адреса (например, идентификатор 0x0020AF определяет компанию 3COM, а 0x00000C - Cisco). За уникальность младших трех байтов адреса отвечает производитель оборудования. Двадцать четыре бита, отводимые производителю для адресации интерфейсов его продукции, позволяют выпустить примерно 16 миллионов интерфейсов под одним идентификатором организации. Уникальность централизованно распределяемых адресов распространяется на все основные технологии локальных сетей – Ethernet, Token Ring, FDDI и т. д. Локальные адреса назначаются администратором сети, в обязанности которого входит обеспечение их уникальности.

Сетевые адаптеры Ethernet могут также работать в так называемом «неразборчивом» режиме (promiscuous mode), когда они захватывают все кадры, поступающие на интерфейс, независимо от их MAC-адресов назначения. Обычно такой режим используется для мониторинга трафика, когда захваченные кадры изучаются затем для нахождения причины некорректного поведения некоторого узла или отладки нового протокола.

11.4.2 Форматы кадров технологии Ethernet

Существует несколько стандартов формата кадра Ethernet. На практике в оборудовании Ethernet используется только один формат кадра, а именно - кадр Ethernet DIX, который иногда называют кадром Ethernet II по номеру последнего стандарта DIX. Этот формат представлен на рисунке 2.24.

6 байт	6 байт	2 байта	46–1500 байт	4 байта
DA	SA	T	Данные	FCS

Рисунок 2.24 – Формат кадра Ethernet DIX (II)

Первые два поля заголовка отведены под адреса:

- *DA (Destination Address)* — MAC-адрес узла назначения;

- *SA (Source Address*) — MAC-адрес узла отправителя.

Для доставки кадра достаточно одного адреса — адреса назначения; адрес источника помещается в кадр для того, чтобы узел, получивший кадр, знал, от кого пришел кадр и кому нужно на него ответить. Принятие решения об ответе не входит в компетенцию протокола Ethernet, это дело протоколов верхних уровней, Ethernet лишь выполнит такое действие, если с сетевого уровня поступит соответствующее указание.

Поле T (Type, или EtherType) содержит условный код протокола верхнего уровня, данные которого находятся в поле данных кадра, например, шестнадцатеричное значение 08-00 соответствует протоколу IP. Это поле требуется для поддержки интерфейсных функций мультиплексирования и демультиплексирования кадров при взаимодействии с протоколами верхних уровней.

Поле данных может содержать от 46 до 1500 байт. Если длина пользовательских данных меньше 46 байт, то это поле дополняется до минимального размера байтами заполнения. Эта операция требуется для корректной работы метода доступа Ethernet (он рассматривается в следующем разделе).

Поле контрольной последовательности кадра (Frame Check Sequence, FCS) состоит из 4 байт контрольной суммы. Это значение вычисляется по алгоритму CRC-32.

Кадр Ethernet DIX (II) не отражает разделения канального уровня Ethernet на уровни MAC и LLC: его поля поддерживают функции обоих уровней, например, интерфейсные функции поля T относятся к функциям уровня LLC, в то время как все остальные поля поддерживают функции уровня MAC.

Существуют еще три стандартных формата кадра Ethernet:

- *кадр 802.3/LLC* является стандартом комитета IEEE 802 и построен в соответствии с принятым разбиением функций канального уровня на уровни MAC и LLC. Поэтому результирующий кадр является вложением кадра LLC, определяемого стандартом 802.2, в кадр MAC, определяемый стандартом 802.3;

- *кадр Raw 802.3, или Novell 802.3,* появился в результате усилий компании Novell по ускорению разработки своего стека протоколов в сетях Ethernet;

- *кадр Ethernet SNAP* стал результатом деятельности комитета 802.2 по приведению предыдущих форматов кадров к некоторому общему стандарту и приданию кадру необходимой гибкости для учета в будущем возможностей добавления полей или изменения их назначения.

Как уже отмечалось, в настоящее время оборудованием Ethernet используются только кадры Ethernet DIX (II). Остальные форматы кадров, в том

числе кадр 802.3/LLC, по- прежнему формально являющийся стандартным, вышли из употребления из-за более сложного формата, который оказался не нужен в условиях существования единой технологии канального уровня.

11.3.3 Метод доступа CSMA/CD

В сетях Ethernet используется метод доступа к среде передачи данных, называемый методом коллективного доступа с опознаванием несущей и обнаружением коллизий (carrier-sense-multiply-access with collision detection, CSMA/CD). Этот метод применяется исключительно в сетях с логической общей шиной (к которым относятся и радиосети, породившие этот метод). Все компьютеры такой сети имеют непосредственный доступ к общей шине, поэтому она может быть использована для передачи данных между любыми двумя узлами сети. Одновременно все компьютеры сети имеют возможность немедленно (с учетом задержки распространения сигнала по физической среде) получить данные, которые любой из компьютеров начал передавать на общую шину (рисунок 2.25). Простота схемы подключения — это один из факторов, определивших успех стандарта Ethernet. Говорят, что кабель, к которому подключены все станции, работает в режиме коллективного доступа (Multiply Access, MA).

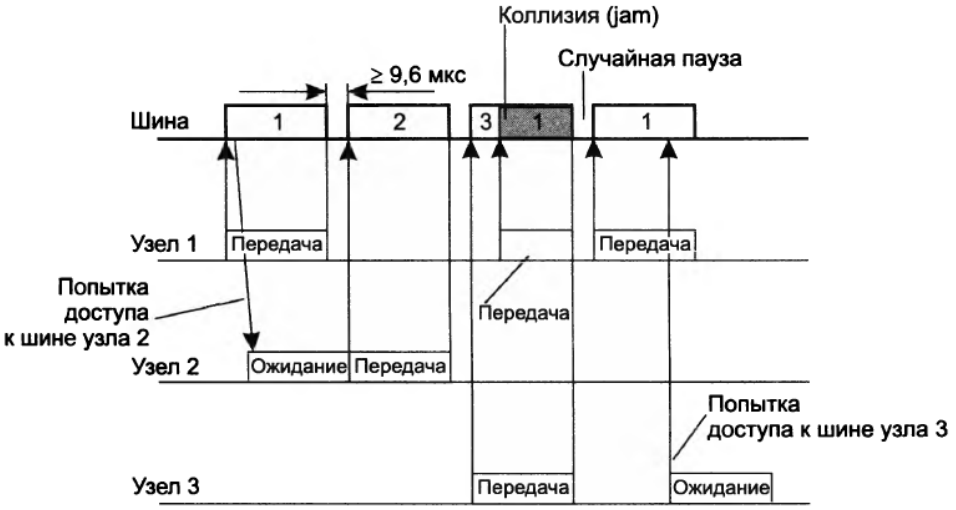


Рисунок 2.25 – Метод случайного доступа CSMA/CD

Этапы доступа к среде. Все данные, передаваемые по сети, помещаются в кадры определенной структуры и снабжаются уникальным адресом станции назначения.

Чтобы получить возможность передавать кадр, станция должна убедиться, что разделяемая среда свободна. Это достигается прослушиванием основной гармоники сигнала, которая также называется несущей частотой (carrier-sense, CS). Признаком незанятости среды является отсутствие на ней несущей частоты,

которая при манчестерском способе кодирования равна 5-10 МГц, в зависимости от последовательности единиц и нулей, передаваемых в данный момент.

Если среда свободна, то узел имеет право начать передачу кадра. Этот кадр изображен на рисунке 2.25 первым. Узел 1 обнаружил, что среда свободна, и начал передавать свой кадр. В классической сети Ethernet на коаксиальном кабеле сигналы передатчика узла 1 распространяются в обе стороны, так что все узлы сети их получают. Кадр данных всегда сопровождается преамбулой (preamble), которая состоит из 7 байт, состоящих из значений 10101010, и 8-го байта, равного 10101011. Преамбула нужна для вхождения приемника в побитовый и побайтовый синхронизм с передатчиком.

Все станции, подключенные к кабелю, могут распознать факт передачи кадра, и та станция, которая узнает собственный адрес в заголовках кадра, записывает его содержимое в свой внутренний буфер, обрабатывает полученные данные, передает их вверх по своему стеку, а затем посылает по кабелю кадр-ответ. Адрес станции-источника содержится в исходном кадре, поэтому станция-получатель знает, кому нужно послать ответ.

Узел 2 во время передачи кадра узлом 1 также пытался начать передачу своего кадра, однако обнаружил, что среда занята — на ней присутствует несущая частота, — поэтому узел 2 вынужден ждать, пока узел 1 не прекратит передачу кадра.

После окончания передачи кадра все узлы сети обязаны выдержать технологическую паузу (Inter Packet Gap) в 9,6 мкс. Эта пауза, называемая также межкадровым интервалом, нужна для приведения сетевых адаптеров в исходное состояние, а также для предотвращения монопольного захвата среды одной станцией. После окончания технологической паузы узлы имеют право начать передачу своего кадра, так как среда свободна. Из-за задержек распространения сигнала по кабелю не все узлы строго одновременно фиксируют факт окончания передачи кадра узлом 1.

В приведенном примере узел 2 дождался окончания передачи кадра узлом 1, сделал паузу в 9,6 мкс и начал передачу своего кадра.

Возникновение коллизии. При описанном подходе возможна ситуация, когда две станции одновременно пытаются передать кадр данных по общей среде. Механизм прослушивания среды и пауза между кадрами не гарантируют от возникновения такой ситуации, когда две или более станции одновременно решают, что среда свободна, и начинают передавать свои кадры. Говорят, что при этом происходит коллизия (collision), так как содержимое обоих кадров сталкивается на общем кабеле и происходит искажение информации — методы кодирования, используемые в Ethernet, не позволяют выделять сигналы каждой станции из общего сигнала.

Коллизия — это нормальная ситуация в работе сетей Ethernet. В примере, изображенном на рисунке 2.26, коллизию породила одновременная передача данных узлами 3 и 1.

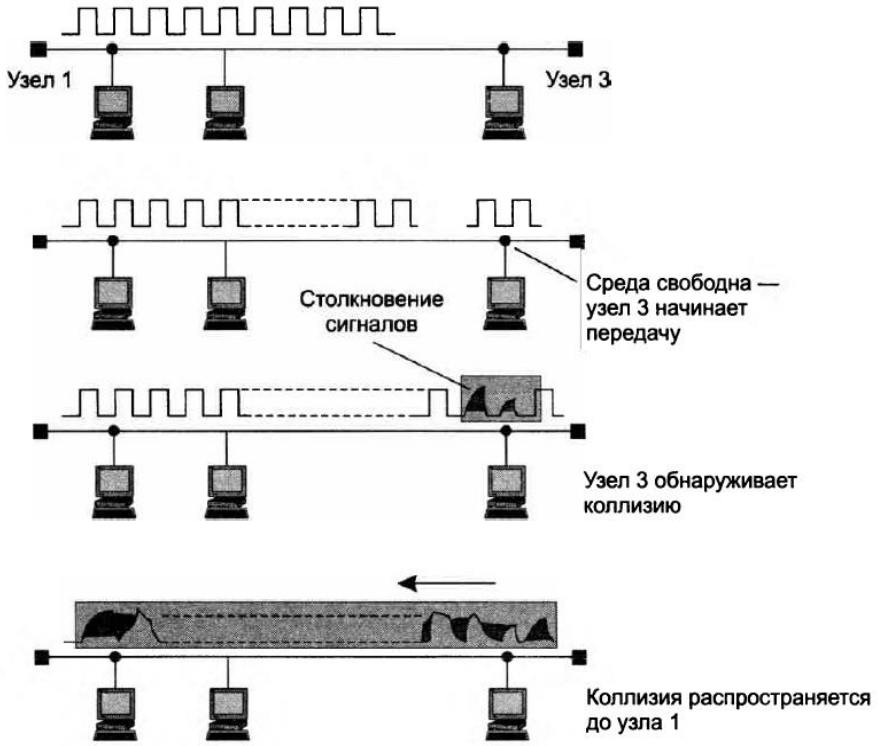


Рисунок 2.26 – Схема возникновения и распространения коллизии

Чтобы корректно обработать коллизию, все станции одновременно наблюдают за возникающими на кабеле сигналами. Если передаваемые и наблюдаемые сигналы отличаются, то фиксируется обнаружение коллизии (collision detection, CD). Для увеличения вероятности скорейшего обнаружения коллизии всеми станциями сети станция, которая обнаружила коллизию, прерывает передачу своего кадра (в произвольном месте, возможно, и не на границе байта) и усиливает ситуацию коллизии посылкой в сеть специальной последовательности из 32 бит, называемой jam-последовательностью.

После этого обнаружившая коллизию передающая станция обязана прекратить передачу и сделать паузу в течение короткого случайного интервала времени. Затем она может снова предпринять попытку захвата среды и передачи кадра. Случайная пауза выбирается по следующему алгоритму:

Пауза = L x (интервал отсрочки),

где интервал отсрочки равен 512 битовым интервалам (в технологии Ethernet принято все интервалы измерять в битовых интервалах; битовый интервал обозначается как *bt* и соответствует времени между появлением двух последовательных бит данных на кабеле; для скорости 10 Мбит/с величина битового интервала равна 0,1 мкс или 100 нс);

L представляет собой целое число, выбранное с равной вероятностью из диапазона [0, 2^N], где N — номер повторной попытки передачи данного кадра: 1,2,..., 10.

После 10-й попытки интервал, из которого выбирается пауза, не увеличивается. Таким образом, случайная пауза может принимать значения от 0 до 52,4 мс.

Если 16 последовательных попыток передачи кадра вызывают коллизию, то передатчик должен прекратить попытки и отбросить этот кадр.

Из описания метода доступа видно, что он носит вероятностный характер, и вероятность успешного получения в свое распоряжение общей среды зависит от загруженности сети, то есть от интенсивности возникновения в станциях потребности в передаче кадров. При разработке этого метода в конце 70-х годов предполагалось, что скорость передачи данных в 10 Мбит/с очень высока по сравнению с потребностями компьютеров во взаимном обмене данными, поэтому загрузка сети будет всегда небольшой. Это предположение остается иногда справедливым и по сей день, однако уже появились приложения, работающие в реальном масштабе времени с мультимедийной информацией, которые очень загружают сегменты Ethernet. При этом коллизии возникают гораздо чаще. При значительной интенсивности коллизий полезная пропускная способность сети Ethernet резко падает, так как сеть почти постоянно занята повторными попытками передачи кадров. Для уменьшения интенсивности возникновения коллизий нужно либо уменьшить график, сократив, например, количество узлов в сегменте или заменив приложения, либо повысить скорость протокола, например, перейти на Fast Ethernet.

Следует отметить, что метод доступа CSMA/CD вообще не гарантирует станции, что она когда-либо сможет получить доступ к среде. Конечно, при небольшой загрузке сети вероятность такого события невелика, но при коэффициенте использования сети, приближающемся к 1, такое событие становится очень вероятным. Этот недостаток метода случайного доступа — плата за его чрезвычайную простоту, которая сделала технологию Ethernet самой недорогой. Другие методы доступа — маркерный доступ сетей Token Ring и FDDI, метод Demand Priority сетей 100VG-AnyLAN — свободны от этого недостатка.

11.3.4 Время двойного оборота и распознавание коллизий

Четкое распознавание коллизий всеми станциями сети является необходимым условием корректной работы сети Ethernet. Если какая-либо передающая станция не распознает коллизию и решит, что кадр данных ею передан верно, то этот кадр данных будет утерян. Из-за наложения сигналов при коллизии информация кадра исказится, и он будет отбракован принимающей станцией (возможно, из-за несовпадения контрольной суммы). Скорее всего,

искаженная информация будет повторно передана каким-либо протоколом верхнего уровня, например, транспортным или прикладным, работающим с установлением соединения. Но повторная передача сообщения протоколами верхних уровней произойдет через значительно более длительный интервал времени (иногда даже через несколько секунд) по сравнению с микросекундными интервалами, которыми оперирует протокол Ethernet. Поэтому если коллизии не будут надежно распознаваться узлами сети Ethernet, то это приведет к заметному снижению полезной пропускной способности данной сети.

Для надежного распознавания коллизий должно выполняться следующее соотношение:

$T_{min} >= PDV,$

где T_{min} — время передачи кадра минимальной длины, а PDV — время, за которое сигнал коллизии успевает распространиться до самого дальнего узла сети. Так как в худшем случае сигнал должен пройти дважды между наиболее удаленными друг от друга станциями сети (в одну сторону проходит неискаженный сигнал, а на обратном пути распространяется уже искаженный коллизией сигнал), то это время называется временем двойного оборота (Path Delay Value, PDV).

При выполнении этого условия передающая станция должна успевать обнаружить коллизию, которую вызвал переданный ее кадр, еще до того, как она закончит передачу этого кадра.

Очевидно, что выполнение этого условия зависит, с одной стороны, от длины минимального кадра и пропускной способности сети, а с другой стороны, от длины кабельной системы сети и скорости распространения сигнала в кабеле (для разных типов кабеля эта скорость несколько отличается).

Все параметры протокола Ethernet подобраны таким образом, чтобы при нормальной работе узлов сети коллизии всегда четко распознавались. При выборе параметров, конечно, учитывалось и приведенное выше соотношение, связывающее между собой минимальную длину кадра и максимальное расстояние между станциями в сегменте сети.

В стандарте Ethernet принято, что минимальная длина поля данных кадра составляет 46 байт (что вместе со служебными полями дает минимальную длину кадра 64 байт, а вместе с преамбулой — 72 байт или 576 бит). Отсюда может быть определено ограничение на расстояние между станциями.

Итак, в 10-мегабитном Ethernet время передачи кадра минимальной длины равно 575 битовых интервалов, следовательно, время двойного оборота должно быть меньше 57,5 мкс. Расстояние, которое сигнал может пройти за это время, зависит от типа кабеля и для толстого коаксиального кабеля равно примерно 13280 м. Учитывая, что за это время сигнал должен пройти по линии связи дважды, расстояние между двумя узлами не должно быть больше 6635 м. В стандарте

величина этого расстояния выбрана существенно меньше, с учетом других, более строгих ограничений.

Одно из таких ограничений связано с предельно допустимым затуханием сигнала. Для обеспечения необходимой мощности сигнала при его прохождении между наиболее удаленными друг от друга станциями сегмента кабеля максимальная длина непрерывного сегмента толстого коаксиального кабеля с учетом вносимого им затухания выбрана в 500 м. Очевидно, что на кабеле в 500 м условия распознавания коллизий будут выполняться с большим запасом для кадров любой стандартной длины, в том числе и 72 байт (время двойного оборота по кабелю 500 м составляет всего 43,3 битовых интервала). Поэтому минимальная длина кадра могла бы быть установлена еще меньше. Однако разработчики технологии не стали уменьшать минимальную длину кадра, имея в виду многосегментные сети, которые строятся из нескольких сегментов, соединенных повторителями.

Повторители увеличивают мощность передаваемых с сегмента на сегмент сигналов, в результате затухание сигналов уменьшается и можно использовать сеть гораздо большей длины, состоящую из нескольких сегментов. В коаксиальных реализациях Ethernet разработчики ограничили максимальное количество сегментов в сети пятью, что в свою очередь ограничивает общую длину сети 2500 метрами. Даже в такой многосегментной сети условие обнаружения коллизий по-прежнему выполняется с большим запасом (сравним полученное из условия допустимого затухания расстояние в 2500 м с вычисленным выше максимально возможным по времени распространения сигнала расстоянием 6635 м). Однако в действительности временной запас является существенно меньше, поскольку в многосегментных сетях сами повторители вносят в распространение сигнала дополнительную задержку в несколько десятков битовых интервалов. Естественно, небольшой запас был сделан также для компенсации отклонений параметров кабеля и повторителей.

В результате учета всех этих и некоторых других факторов было тщательно подобрано соотношение между минимальной длиной кадра и максимально возможным расстоянием между станциями сети, которое обеспечивает надежное распознавание коллизий. Это расстояние называют также максимальным диаметром сети.

С увеличением скорости передачи кадров, что имеет место в новых стандартах, базирующихся на том же методе доступа CSMA/CD, например, Fast Ethernet, максимальное расстояние между станциями сети уменьшается пропорционально увеличению скорости передачи. В стандарте Fast Ethernet оно составляет около 210 м, а в стандарте Gigabit Ethernet оно было бы ограничено 25 метрами, если бы разработчики стандарта не предприняли некоторых мер по увеличению минимального размера пакета.

11.3.5 Максимальная производительность сети Ethernet

Количество обрабатываемых кадров Ethernet в секунду часто указывается производителями мостов/коммутаторов и маршрутизаторов как основная характеристика производительности этих устройств. В свою очередь, интересно знать чистую максимальную пропускную способность сегмента Ethernet в кадрах в секунду в идеальном случае, когда в сети нет коллизий и нет дополнительных задержек, вносимых мостами и маршрутизаторами. Такой показатель помогает оценить требования к производительности коммуникационных устройств, так как в каждый порт устройства не может поступать больше кадров в единицу времени, чем позволяет это сделать соответствующий протокол.

Для коммуникационного оборудования наиболее тяжелым режимом является обработка кадров минимальной длины. Это объясняется тем, что на обработку каждого кадра мост, коммутатор или маршрутизатор тратит примерно одно и то же время, связанное с просмотром таблицы продвижения пакета, формированием нового кадра (для маршрутизатора) и т. п. А количество кадров минимальной длины, поступающих на устройство в единицу времени, естественно больше, чем кадров любой другой длины. Другая характеристика производительности коммуникационного оборудования — бит в секунду — используется реже, так как она не говорито том, какого размера кадры при этом обрабатывало устройство, а на кадрах максимального размера достичь высокой производительности, измеряемой в битах в секунду гораздо легче.

Используя параметры, приведенные в табл. 3.1, рассчитаем максимальную производительность сегмента Ethernet в таких единицах, как число переданных кадров (пакетов) минимальной длины в секунду

ПРИМЕЧАНИЕ. При указании пропускной способности сетей термины кадр и пакет обычно используются как синонимы. Соответственно, аналогичными являются и единицы измерения производительности frames-per-second, fps и packels-per-second, pps.

Для расчета максимального количества кадров минимальной длины, проходящих по сегменту Ethernet, заметим, что размер кадра минимальной длины вместе с преамбулой составляет 72 байт или 576 бит (рисунок 2.27), поэтому на его передачу затрачивается 57,5 мкс. Прибавив межкадровый интервал в 9,6 мкс, получаем, что период следования кадров минимальной длины составляет 67,1 мкс. Отсюда максимально возможная пропускная способность сегмента Ethernet составляет 14 880 кадр/с.

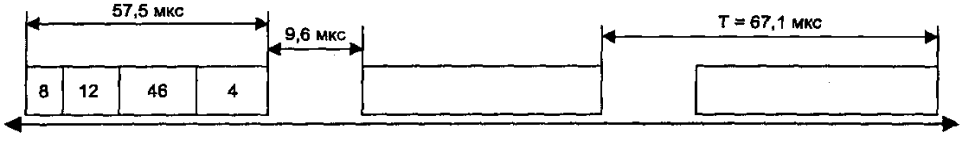


Рисунок 2.27 – К расчету пропускной способности канала

Естественно, что наличие в сегменте нескольких узлов снижает эту величину за счет ожидания доступа к среде, а также за счет коллизий, приводящих к необходимости повторной передачи кадров.

Кадры максимальной длины технологии Ethernet имеют поле длины 1500 байт, что вместе со служебной информацией дает 1518 байт, а с преамбулой составляет 1526 байт или 12 208 бит Максимально возможная пропускная способность сегмента Ethernet для кадров максимальной длины составляет 813 кадр/с. Очевидно, что при работе с большими кадрами нагрузка на мосты, коммутаторы и маршрутизаторы довольно ощутимо снижается.

Теперь рассчитаем, какой максимальной полезной пропускной способностью в бит в секунду обладают сегменты Ethernet при использовании кадров разного размера.

Под *полезной пропускной способностью протокола* понимается скорость передачи пользовательских данных, которые переносятся полем данных кадра. Эта пропускная способность всегда меньше номинальной битовой скорости протокола Ethernet за счет нескольких факторов:

- служебной информации кадра;
- межкадровых интервалов (IPG);
- ожидания доступа к среде.

Для кадров минимальной длины полезная пропускная способность равна:

C = 14880 x 46 x 8 = 5,48 Мбит/с.

Это намного меньше 10 Мбит/с, но следует учесть, что кадры минимальной длины используются в основном для передачи квитанций, так что к передаче собственно данных файлов эта скорость отношения не имеет.

Для кадров максимальной длины полезная пропускная способность равна:

C = 813 x 1500 x 8 x 9,76 Мбит/с,

что весьма близко к номинальной скорости протокола.

Еще раз подчеркнем, что такой скорости можно достигнуть только в том случае, когда двум взаимодействующим узлам в сети Ethernet другие узлы не мешают, что бывает крайне редко.

При использовании кадров среднего размера с полем данных в 512 байт пропускная способность сети составит 9,29 Мбит/с, что тоже достаточно близко к предельной пропускной способности в 10 Мбит/с.

При отсутствии коллизий и ожидания доступа коэффициент использования сети зависит от размера поля данных кадра и имеет максимальное значение 0,976 при передаче кадров максимальной длины. Очевидно, что в реальной сети Ethernet среднее значение коэффициента использования сети может значительно отличаться от этой величины. Более сложные случаи определения пропускной

способности сети с учетом ожидания доступа и отработки коллизий будут рассмотрены ниже.

11.3.6 Физические стандарты 10M Ethernet

При первоначальной стандартизации технологии Ethernet рабочей группой IEEE 802.3 был выбран вариант Ethernet на <толстом> коаксиальном кабеле, который получил название 10Base-5.

Число 10 этом названии обозначает номинальную битовую скорость передачи данных стандарта, то есть 10 Мбит/с, а слово «Base» – метод передачи на одной базовой частоте (в данном случае - 10 МГц). Последний символ в названии стандарта физического уровня обозначает тип кабеля, в данном случае 5 отражает тот факт, что диаметр «толстого» коаксиала равен 0,5 дюйма. Данный подход к обозначению типа физического уровня Ethernet сохранился до настоящего времени, только вместо диаметра коаксиального кабеля в современных стандартах кодируется тип кабеля (например, 1000Base-T определяет спецификацию для витой пары) или же способ кодирования данных.

В качества метода кодирования сигналов был выбран манчестерский код.

Затем сети Ethernet на «толстом» коаксиальном кабеле были вытеснены сетями на более «тонком» коаксиале (диаметром 0,25 дюйма, что отражает название 10Base-2 этого стандарта), который позволял строить сети более экономичным способом.

Однако сети Ethernet на коаксиальном кабеле обладали одним существенным недостатком, а именно отсутствием оперативной информации о состоянии кабеля и сложностью нахождения места его повреждения. Поэтому поиск неисправностей стал привычной процедурой и головной болью многочисленной армии сетевых администраторов коаксиальных сетей Ethernet.

Альтернатива появилась в середине 80-х годов, когда благодаря использованию витой пары и повторителей сети Ethernet стали гораздо более ремонтопригодными.

К этому времени телефонные компании уже достаточно давно применяли многопарный кабель на основе неэкранированной витой пары для подключения телефонных аппаратов внутри зданий. Идея приспособить этот популярный вид кабеля для локальных сетей оказалась очень плодотворной, так как многие здания уже были оснащены нужной кабельной системой. Оставалось разработать способ подключения сетевых адаптеров и прочего коммуникационного оборудования к витой паре таким образом, чтобы изменения в сетевых адаптерах и программном обеспечении сетевых операционных систем были минимальными по сравнению с сетями Ethernet на коаксиале. Эта попытка оказалась успешной – переход на витую пару требует только замены приемника и передатчика сетевого адаптера, а метод доступа и все протоколы канального уровня остаются теми же, что и в сетях

Ethernet на коаксиале. Результатом реализации этой идеи стал стандарт 10Base-T (T – от Twisted pair).

Правда, для соединения узлов в сеть теперь обязательно требуется коммуникационное устройство – многопортовый повторитель Ethernet на витой паре.

Устройство такого повторителя схематично изображено на рисунке 2.28. Каждый сетевой адаптер соединяется с повторителем двумя витыми парами. Одна витая пара требуется для передачи данных от станции к повторителю (выход Tx сетевого адаптера), другая – для передачи данных от повторителя к станции (вход Rx сетевого адаптера). Повторитель побитно принимает сигналы от одного из конечных узлов и синхронно передает их на все свои остальные порты, исключая тот, с которого поступили сигналы, одновременно улучшая их электрические характеристики.

Многопортовый повторитель часто называют концентратором, или хабом (от английского hub - центр, ступица колеса), так как в нем сконцентрированы соединения со всеми конечными узлами сети. Фактически хаб имитирует сеть на коаксиальном кабеле в том отношении, что физически отдельные отрезки кабеля на витой паре логически все равно представляют единую разделяемую среду. Все правила доступа к среде по алгоритму CSMA/CD сохраняются.

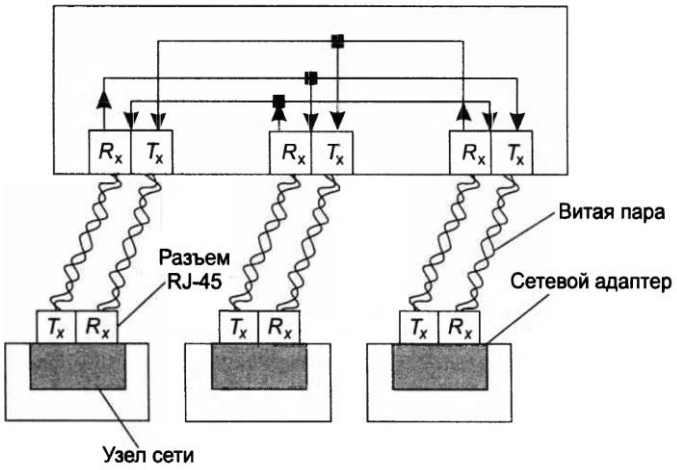


Рисунок 2.28 – Повторитель Ethernet на витой паре

При создании сети Ethernet на витой паре с большим числом конечных узлов хабы можно соединять друг с другом иерархическим способом, образуя древовидную структуру (рисунок 2.29). Добавление каждого хаба изменяет физическую структуру, но оставляет без изменения логическую структуру сети. То есть независимо от числа хабов в сети сохраняется одна общая для всех интерфейсов разделяемая среда, так что передача кадра с любого интерфейса блокирует передатчики всех остальных интерфейсов.

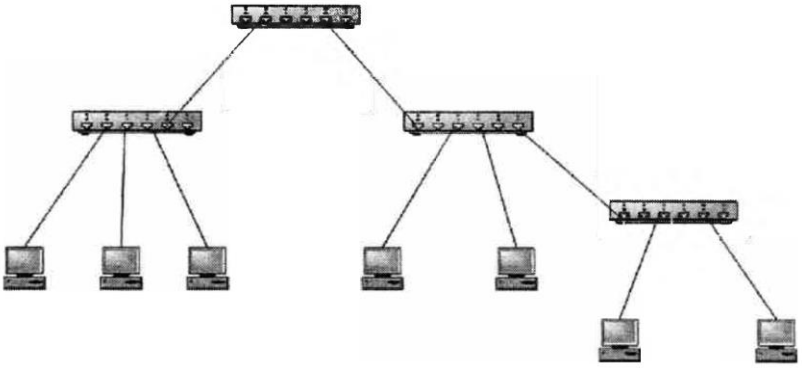


Рисунок 2.29 – Иерархическое соединение хабов

Физическая структуризация сетей, построенных на основе витой пары, повышает надежность и упрощает обслуживание сети, поскольку в этом случае появляется возможность контролировать состояние и локализовать отказы отдельных кабельных отрезков, подключающих конечные узлы к концентраторам. В случае обрыва, короткого замыкания или неисправности сетевого адаптера работа сети может быть быстро восстановлена путем отключения соответствующего сегмента кабеля.

Для контроля целостности физического соединения между двумя непосредственно соединенными портами в стандарте 10Base-T введен так называемый тест целостности соединения (Link Integrity Test, LIT). Эта процедура заключается в том, что в те периоды, когда порт не посылает или не получает кадры данных, он посылает своему соседу импульсы длительностью 100 не через каждые 16 мс. Если порт принимает такие импульсы от своего соседа, то он считает соединение работоспособным и, как правило, индицирует это зеленым светом светодиода.

Вскоре появился стандарт 10Base-F, в котором вместо витых пар используется оптическое волокно, обеспечивающее большее расстояние между узлом сети и повторителем.

Во всех вариантах физической среды Ethernet со скоростью 10 Мбит/с используется манчестерский код.

11.4 Коммутируемые сети Ethernet

Мост как предшественник и функциональный аналог коммутатора. Современные коммутаторы Ethernet являются наследниками мостов локальных сетей, которые широко использовались в сетях Ethernet и Token Ring на разделяемой среде. Более того, коммутаторы Ethernet по-прежнему функционально очень близки к вышедшим из употребления мостам, так как базовый алгоритм работы коммутатора и моста является одним и тем же и

определяется одним стандартом IEEE 802.ID, хотя по традиции во всех новых стандартах IEEE, описывающих свойства коммутаторов, употребляется термин ≪коммутатор≫, а не ≪мост≫. Основное отличие коммутатора от моста состоит в большем количестве портов (мост, как правило, имел два порта, что и послужило поводом для его названия — мост между двумя сегментами) и более высокой производительности. Далее мы будем использовать эти термины как синонимы, выбирая нужный в зависимости от контекста.

Коммутаторы являются сегодня основным типом коммуникационных устройств, применяемых для построения локальных сетей. Коммутаторы отличаются внутренней архитектурой и конструктивным исполнением.

Коммутируемые локальные сети вытеснили локальные сети на разделяемой среде. Их успех оказал решающее влияние на эволюцию Ethernet — в новых скоростных версиях Ethernet, таких как 10G и 100G Ethernet, стандарт IEEE 802.3 описывает работу узлов только в коммутируемой среде.

11.4.1 Логическая структуризация сетей и мосты

Мост локальной сети (LAN bridge), или просто мост, появился как средство построения крупных локальных сетей на разделяемой среде. Мост объединяет две или более разделяемые среды в единую сеть, при этом передача кадров между узлами каждой из объединяемых сред происходит по стандартным правилам изолированной разделяемой среды. Мост отвечает только за передачу кадров между объединенными средами, которые называются сегментами локальной сети.

В сети Ethernet требование использовать единую разделяемую среду приводит к двум очень жестким ограничениям:

- общий диаметр сети не может быть больше 2500 м;
- количество узлов не может превышать 1024 (для сетей Ethernet на коаксиале это ограничение еще жестче).

Качественная картина зависимости задержек доступа к разделяемой среде от коэффициента использования среды показана на рисунке 2.30.

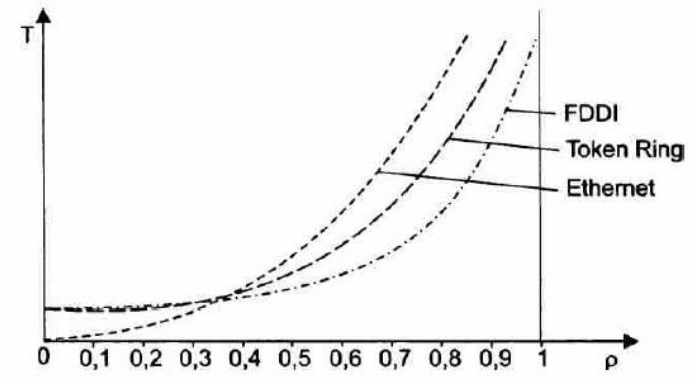


Рисунок 2.30 – Зависимость задержек доступа к разделяемой среде от коэффициента использования среды

Как видно из рисунка, всем технологиям присуща качественно одинаковая картина экспоненциального роста величины задержек доступа при увеличении коэффициента использования сети. Количество узлов, при которых коэффициент использования сети начинает приближаться к опасной границе, зависит от типа функционирующих в узлах приложений. Для сетей Ethernet со скоростью 10 Мбит/с считалось, что 30 узлов — это вполне приемлемое число для одного разделяемого сегмента, так что для построения крупной сети нужны были принципиально новые решения.

Ограничения, возникающие из-за использования единой разделяемой среды, можно преодолеть, выполнив логическую структуризацию сети, то есть сегментировав единую разделяемую среду на несколько и соединив полученные сегменты сети некоторым коммуникационным устройством, которое не передает данные побитно, как повторитель, а буферизует кадры и передает их затем в тот или иной сегмент (или сегменты) в зависимости от адреса назначения кадра (рисунок 2.31).

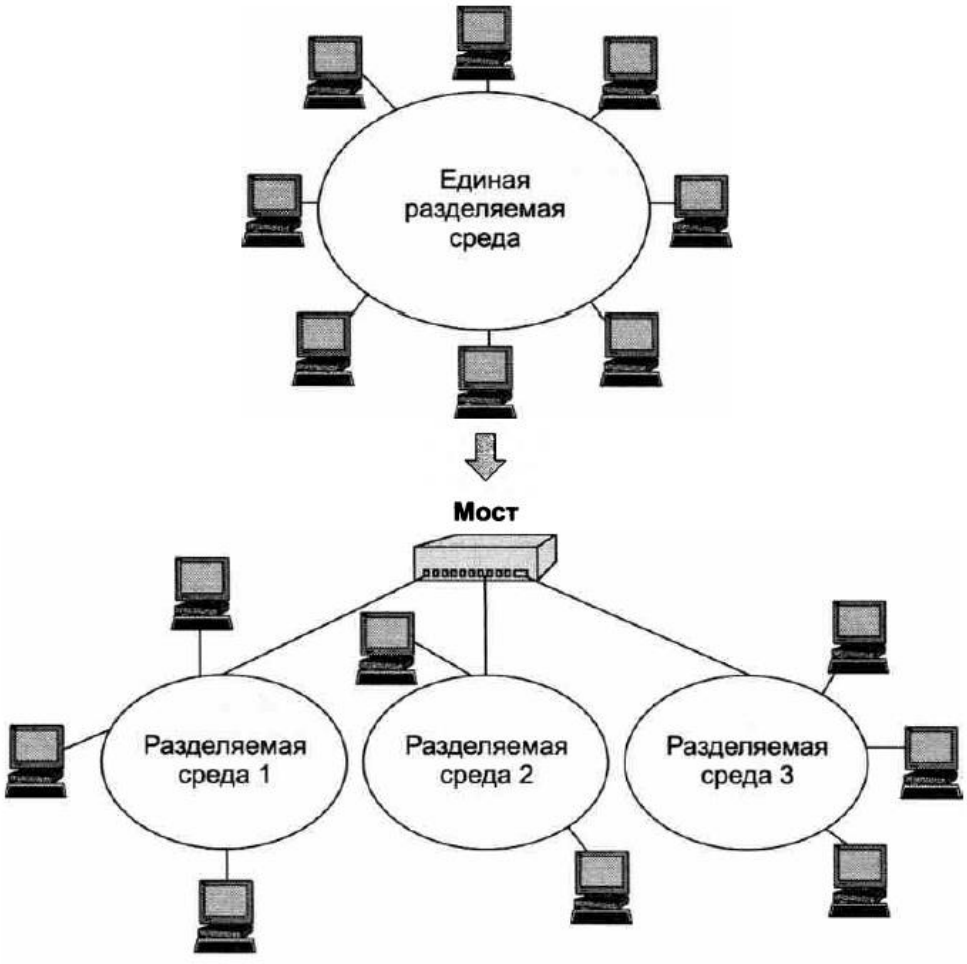


Рисунок 2.31 – К вопросу о логической структуризации сети

Нужно отличать логическую структуризацию от физической. Например, концентраторы стандарта 10Base-T позволяют построить сеть, состоящую из нескольких сегментов кабеля на витой паре, но это — физическая структуризация, так как логически все эти сегменты представляют собой единую разделяемую среду. Логическая структуризация сети с помощью мостов/коммутаторов является первым шагом на пути виртуализации сети, так как пользователям отдельного логического сегмента предоставляется виртуальный ресурс — коммуникационная среда с определенной пропускной способностью.

Логическая структуризация локальной сети позволяет решить несколько задач, основные из которых — повышение производительности, гибкости и безопасности, а также улучшение управляемости сети.Повышение производительности сети, разделенной мостом на сегменты, происходит из-за того, что среда каждого сегмента разделяется теперь между меньшим числом конечных узлов. В примере на рисунке 2.31 при разделении общей среды на три сегмента максимальное количество узлов, разделяющих среду, снизилось с 8 до 3. Как правило, разбиение на сегменты выполняется так, чтобы межсегментный трафик был небольшим, этого можно добиться, например, если каждый сегмент снабдить собственным сервером, обслуживающим запросы компьютеров сегмента.

При построении сети как совокупности сегментов каждый из них может быть адаптирован к специфическим потребностям рабочей группы или отдела. Это означает повышение гибкости сети. Процесс разбиения сети на логические сегменты можно рассматривать и в обратном направлении, как процесс создания большой сети из уже имеющихся небольших сетей.

Устанавливая различные логические фильтры на мостах/коммутаторах, можно контролировать доступ пользователей к ресурсам других сегментов, чего не позволяют делать повторители. Так достигается повышение безопасности данных.

Побочным эффектом снижения трафика и повышения безопасности данных является упрощение управления сетью, то есть улучшение управляемости сети. Проблемы очень часто локализуются внутри сегмента. Сегменты образуют логические домены управления сетью.

Как мосты, так и коммутаторы продвигают кадры на основании одного и того же алгоритма, а именно алгоритма прозрачного моста, описанного в стандарте IEEE 802.ID.

11.4.2 Алгоритм прозрачного моста IEEE 802.1D

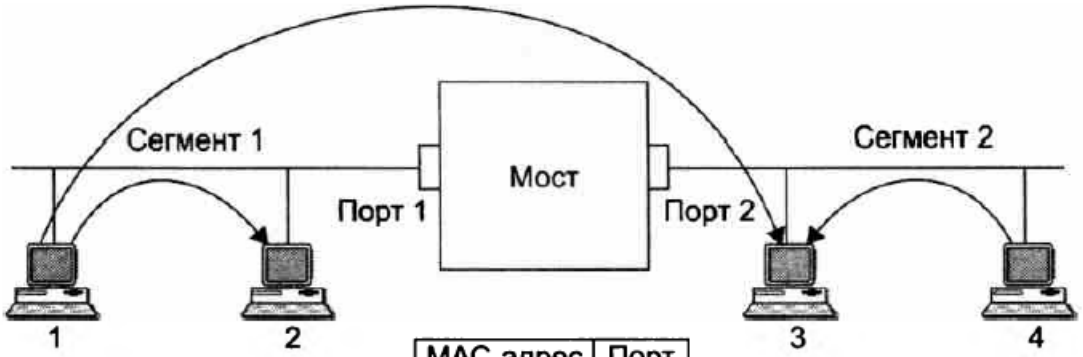
Слово «прозрачный» в названии алгоритм прозрачного моста отражает тот факт, что конечные узлы сети функционируют, <не замечая> присутствия в сети мостов.

Так как алгоритм прозрачного моста остался единственным актуальным алгоритмом мостов, то в дальнейшем мы будем опускать термин «прозрачный», подразумевая именно этот тип алгоритма работы моста/коммутатора.

Мост строит свою таблицу продвижения (адресную таблицу) на основании пассивного наблюдения за трафиком, циркулирующим в подключенных к его портам сегментах. При этом мост учитывает адреса источников кадров данных, поступающих на его порты. По адресу источника кадра мост делает вывод о принадлежности узла-источника тому или иному сегменту сети.

Каждый порт моста работает как конечный узел своего сегмента, за одним исключением — порт моста может не иметь собственного M AC-адреса. Порты мостов не нуждаются в адресах для продвижения кадров, так как они работают в неразборчивом режиме захвата кадров, когда все поступающие на порт кадры независимо от их адреса назначения запоминаются на время в буферной памяти. Работая в неразборчивом режиме, мост «слушает» весь трафик, передаваемый в присоединенных к нему сегментах, и использует проходящие через него кадры для изучения топологии сети и построения таблицы продвижения. В том случае, когда порт моста/коммутатора имеет собственный MAC-адрес, он используется для целей, отличных от продвижения кадров, чаще всего — для удаленного управления портом; в этом случае порт представляет собой конечный узел сети и кадры протокола управления адресуются непосредственно ему.

Рассмотрим процесс автоматического создания таблицы продвижения моста и ее использования на примере простой сети, представленной на рисунке 2.32.



MAC-адрес	Порт
1	1
2	1
3	2
4	2

Рисунок 2.32 – Принцип работы прозрачного моста/коммутатора

Мост соединяет два сетевых сегмента. Сегмент 1 составляют компьютеры, подключенные с помощью одного отрезка коаксиального кабеля к порту 1 моста, а сегмент 2 - компьютеры, подключенные с помощью другого отрезка коаксиального кабеля к порту 2 моста. В исходном состоянии мост не знает о том,

компьютеры с какими MAC-адресами подключены к каждому из его портов. В этой ситуации мост просто передает любой захваченный и буферизованный кадр на все свои порты, за исключением того порта, от которого этот кадр получен. В нашем примере у моста только два порта, поэтому он передает кадры с порта 1 на порт 2, и наоборот. Отличие работы моста в этом режиме от повторителя заключается в том, что он передает кадр, предварительно буферизуя его, а не бит за битом, как это делает повторитель. Буферизация отменяет логику работы всех сегментов как единой разделяемой среды. Когда мост собирается передать кадр с сегмента на сегмент, например с сегмента 1 на сегмент 2, он, как обычный конечный узел, пытается получить доступ к разделяемой среде сегмента 2 по правилам алгоритма доступа, в данном примере - по правилам алгоритма CSMA/CD.

Одновременно с передачей кадра на все порты мост изучает адрес источника кадра и делает запись о его принадлежности к тому или иному сегменту в своей адресной таблице. Эту таблицу также называют таблицей фильтрации, или продвижения. Например, получив на порт 1 кадр от компьютера 1, мост делает первую запись в своей адресной таблице:

MAC-адрес 1 - порт 1.

Эта запись означает, что компьютер, имеющий MAC-адрес 1, принадлежит сегменту, подключенному к порту 1 коммутатора. Если все четыре компьютера данной сети проявляют активность и посылают друг другу кадры, то скоро мост построит полную адресную таблицу сети, состоящую из четырех записей - по одной записи на узел (см. рисунок 2.32).

При каждом поступлении кадра на порт моста он прежде всего пытается найти адрес назначения кадра в адресной таблице. Продолжим рассмотрение действий моста на примере (см. рисунок 2.32).

1. При получении кадра, направленного от компьютера 1 компьютеру 3, мост просматривает адресную таблицу на предмет совпадения адреса в какой-либо из ее записей с адресом назначения - MAC-адресом 3. Запись с искомым адресом имеется в адресной таблице.

2. Мост выполняет второй этап анализа таблицы - проверяет, находятся ли компьютеры с адресами источника и назначения в одном сегменте. В примере компьютер 1 (MAC-адрес 1) и компьютер 3 (MAC-адрес 3) находятся в разных сегментах. Следовательно, мост выполняет операцию продвижения (forwarding) кадра - передает кадр на порт 2, ведущий в сегмент получателя, получает доступ к сегменту и передает туда кадр.

3. Если бы оказалось, что компьютеры принадлежали одному сегменту, то кадр просто был бы удален из буфера. Такая операция называется фильтрацией (filtering).

4. Если бы запись о MAC-адресе 3 отсутствовала в адресной таблице, то есть, другими словами, адрес назначения был неизвестен мосту, то он передал бы кадр на все свои порты, кроме порта - источника кадра, как и на начальной стадии процесса обучения.

Процесс обучения моста никогда не заканчивается и происходит одновременно с продвижением и фильтрацией кадров. Мост постоянно следит за адресами источника буферизуемых кадров, чтобы автоматически приспосабливаться к изменениям, происходящим в сети, - перемещениям компьютеров из одного сегмента сети в другой, отключению и появлению новых компьютеров.

Входы адресной таблицы могут быть динамическими, создаваемыми в процессе самообучения моста, и статическими, создаваемыми вручную администратором сети. Статические записи не имеют срока жизни, что дает администратору возможность влиять на работу моста, например, ограничивая передачу кадров с определенными адресами из одного сегмента в другой.

Динамические записи имеют срок жизни - при создании или обновлении записи в адресной таблице с ней связывается отметка времени. По истечении определенного тайм-аута запись помечается как недействительная, если за это время мост не принял ни одного кадра с данным адресом в поле адреса источника. Это дает возможность мосту автоматически реагировать на перемещения компьютера из сегмента в сегмент - при его отключении от старого сегмента запись о принадлежности компьютера к этому сегменту со временем вычеркивается из адресной таблицы. После подключения компьютера к другому сегменту его кадры начнут попадать в буфер моста через другой порт и в адресной таблице появится новая запись, соответствующая текущему состоянию сети.

Кадры с широковещательными и групповыми MAC-адресами, как и кадры с неизвестными адресами назначения, передаются мостом на все его порты. Такой режим распространения кадров называется затоплением сети (flooding). Наличие мостов в сети не препятствует распространению широковещательных и групповых кадров по всем сегментам сети. Однако это является достоинством только тогда, когда такой адрес выработан корректно работающим узлом.

Нередко в результате каких-либо программных или аппаратных сбоев протокол верхнего уровня или сетевой адаптер начинает работать некорректно, а именно постоянно с высокой интенсивностью генерировать кадры с широковещательным адресом. Мост в соответствии со своим алгоритмом передает ошибочный трафик во все сегменты. Такая ситуация называется широковещательным штормом (broadcast storm).

К сожалению, мосты не защищают сети от широковещательного шторма, во всяком случае, по умолчанию, как это делают маршрутизаторы (вы познакомитесь с этим свойством маршрутизаторов в части IV). Максимум, что может сделать

администратор с помощью коммутатора для борьбы с широковещательным штормом, - установить для каждого порта моста предельно допустимую интенсивность передачи кадров с широковещательным адресом. Но при этом нужно точно знать, какая интенсивность является нормальной, а какая - ошибочной. При смене протоколов ситуация в сети может измениться, и то, что вчера считалось ошибочным, сегодня может оказаться нормой.

На рисунке 2.33 показана типичная структура моста. Функции доступа к среде при приеме и передаче кадров выполняют микросхемы MAC, которые идентичны микросхемам сетевого адаптера.

Протокол, реализующий алгоритм коммутатора, располагается между уровнями MAC и LLC.

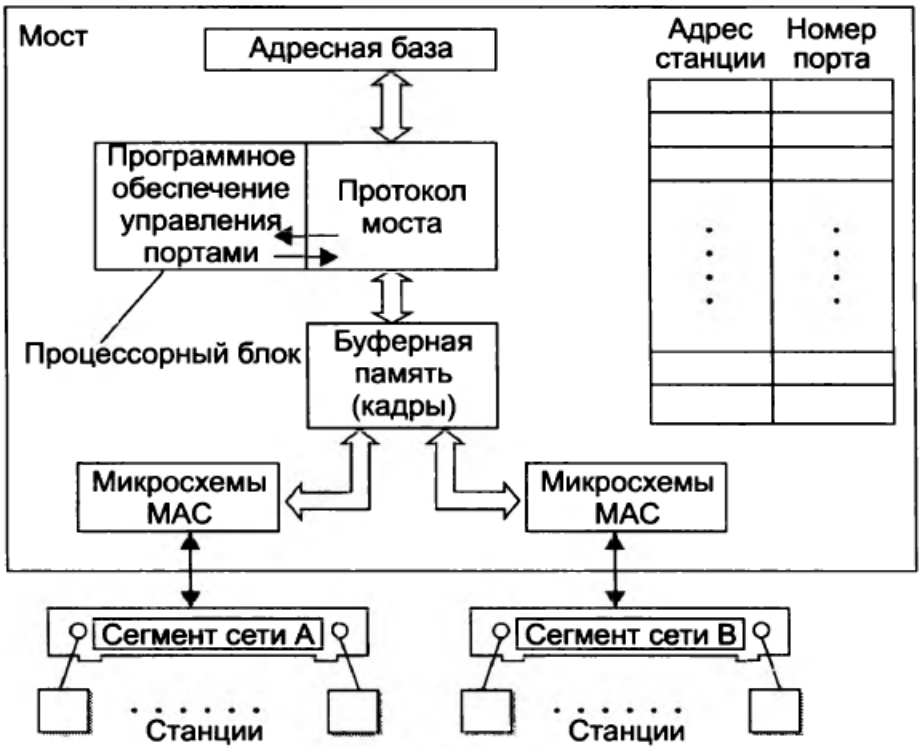


Рисунок 2.33 – Типичная структура моста

11.4.3 Коммутаторы. Параллельная коммутация

При появлении в конце 80-х - начале 90-х годов быстрых протоколов, производительных персональных компьютеров, мультимедийной информации и разделении сети на большое количество сегментов классические мосты перестали справляться с работой. Обслуживание потоков кадров между теперь уже несколькими портами с помощью одного процессорного блока требовало значительного повышения быстродействия процессора, а это довольно дорогостоящее решение.

Более эффективным оказалось решение, которое и <породило> коммутаторы: для обслуживания потока, поступающего на каждый порт, в устройство ставился отдельный специализированный процессор, который реализовывал алгоритм прозрачного моста. По сути, коммутатор — это мультипроцессорный мост, способный параллельно продвигать кадры сразу между всеми парами своих портов. Но если при добавлении процессорных блоков компьютер не перестали называть компьютером, а добавили только прилагательное «мультипроцессорный», то с мультипроцессорными мостами произошла метаморфоза — во многом по маркетинговым причинам они превратились в коммутаторы. Нужно отметить, что помимо процессоров портов коммутатор имеет центральный процессор, который координирует работу портов, отвечая за построение общей таблицы продвижения, а также поддерживая функции конфигурирования и управления коммутатором.

Со временем коммутаторы вытеснили из локальных сетей классические однопроцессорные мосты. Основная причина этого — существенно более высокая производительность, с которой коммутаторы передают кадры между сегментами сети. Если мосты могли даже замедлять работу сети, то коммутаторы всегда выпускаются с процессорами портов, способными передавать кадры с той максимальной скоростью, на которую рассчитан протокол. Ну а добавление к этому возможности параллельной передачи кадров между портами предопределило судьбу и мостов, и коммутаторов.

Производительность коммутаторов на несколько порядков выше, чем мостов, — коммутаторы могут передавать до нескольких десятков, а иногда и сотен миллионов кадров в секунду, в то время как мосты обычно обрабатывали 3-5 тысяч кадров в секунду.

За время своего существования уже без конкурентов-мостов коммутаторы вобрали в себя многие дополнительные функции, родившиеся в результате естественного развития сетевых технологий. К этим функциям относятся, например, поддержка виртуальных сетей (VLAN), агрегирование линий связи, приоритезация трафика и т. п. Развитие технологии производства заказных микросхем также способствовало успеху коммутаторов, в результате процессоры портов сегодня обладают такой вычислительной мощностью, которая позволяет им быстро реализовывать весьма сложные алгоритмы обработки трафика, например, выполнять его классификацию и профилирование.

Основной причиной повышения производительности сети при использовании коммутатора является параллельная обработка нескольких кадров.

11.4.4 Дуплексный режим работы

Технология коммутации сама по себе не имеет непосредственного отношения к методу доступа к среде, который используется портами коммутатора. При

подключении к порту коммутатора сегмента, представляющего собой разделяемую среду, данный порт, как и все остальные узлы такого сегмента, должен поддерживать полудуплексный режим.

Однако когда к каждому порту коммутатора подключен не сегмент, а только один компьютер, причем по двум физически раздельным каналам, как это происходит почти во всех стандартах Ethernet, кроме коаксиальных версий Ethernet, ситуация становится не такой однозначной. Порт может работать как в обычном полудуплексном режиме, так и в дуплексном.

В полудуплексном режиме работы порт коммутатора по-прежнему распознает коллизии. Доменом коллизий в этом случае является участок сети, включающий передатчик коммутатора, приемник коммутатора, передатчик сетевого адаптера компьютера, приемник сетевого адаптера компьютера и две витые пары, соединяющие передатчики с приемниками.

Коллизия возникает, когда передатчики порта коммутатора и сетевого адаптера одновременно или почти одновременно начинают передачу своих кадров.

В дуплексном режиме одновременная передача данных передатчиком порта коммутатора и сетевого адаптера коллизией не считается. В принципе, это достаточно естественный режим работы для отдельных дуплексных каналов передачи данных, и он всегда использовался в протоколах глобальных сетей. При дуплексной связи порты Ethernet стандарта 10 Мбит/с могут передавать данные со скоростью 20 Мбит/с - по 10 Мбит/с в каждом направлении.

Долгое время коммутаторы Ethernet сосуществовали в локальных сетях с концентраторами Ethernet: на концентраторах строились нижние уровни сети здания, такие как сети рабочих групп и отделов, а коммутаторы служили для объединения этих сегментов в общую сеть.

Постепенно коммутаторы стали применяться и на нижних этажах, вытесняя концентраторы, так как цены коммутаторов постоянно снижались, а их производительность росла (за счет поддержки более скоростных версий технологии Ethernet, то есть Fast Ethernet со скоростью 100 Мбит/с, Gigabit Ethernet со скоростью 1 Гбит/с, 10G Ethernet со скоростью 10 Гбит/с и 100G Ethernet со скоростью 100 Гбит/с). Этот процесс завершился вытеснением концентраторов Ethernet и переходом к полностью коммутируемым сетям.

В полностью коммутируемой сети Ethernet все порты работают в дуплексном режиме, а продвижение кадров осуществляется на основе MAC-адресов.

При разработке технологий Fast Ethernet и Gigabit Ethernet дуплексный режим стал одним из двух полноправных стандартных режимов работы узлов сети. Однако практика применения первых коммутаторов с портами Gigabit Ethernet показала, что они практически всегда применяются в дуплексном режиме для взаимодействия с другими коммутаторами или высокоскоростными сетевыми

адаптерами. Поэтому при разработке версий стандартов 10G и 100G Ethernet его разработчики не стали создавать версию для работы в полудуплексном режиме, окончательно закрепив уход разделяемой среды из технологии Ethernet.

11.4.5 Неблокирующие коммутаторы

Как уже отмечалось, высокая производительность является одним из главных достоинств коммутаторов. С понятием производительности тесно связано понятие неблокирующего коммутатора.

Коммутатор называют неблокирующим, если он может передавать кадры через свои порты стой же скоростью, с которой они на них поступают.

Когда говорят, что коммутатор может поддерживать *устойчивый неблокирующий режим работы*, то имеют в виду, что коммутатор передает кадры со скоростью их поступления в течение произвольного промежутка времени. Для поддержания подобного режима нужно таким образом распределить потоки кадров по выходным портам, чтобы, во-первых, порты справлялись с нагрузкой, во-вторых, коммутатор мог всегда в среднем передать на выходы столько кадров, сколько их поступило на входы. Если же входной поток кадров (просуммированный по всем портам) в среднем будет превышать выходной поток кадров (также просуммированный по всем портам), то кадры будут накапливаться в буферной памяти коммутатора и при переполнении — просто отбрасываться.

Для поддержания устойчивого неблокирующего режима работы коммутатора необходимо, чтобы его производительность удовлетворяла условию $C_k = \sum(1C_{pi})/2$, где C_k — производительность коммутатора, C_{pi} — максимальная производительность протокола, поддерживаемого i-м портом коммутатора.

В этом соотношении под производительностью коммутатора в целом понимается его способность продвигать на передатчики всех своих портов определенное количество кадров, принимаемых от приемников всех своих портов.

В суммарной производительности портов каждый проходящий кадр учитывается дважды, как входящий и как выходящий, а так как в устойчивом режиме входной трафик равен выходному, то минимально достаточная производительность коммутатора для поддержки неблокирующего режима равна половине суммарной производительности портов. Если порт, например, стандарта Ethernet со скоростью 10 Мбит/с работает в полудуплексном режиме, то производительность порта C_{pi} равна 10 Мбит/с, а если в дуплексном — 20 Мбит/с.

Иногда говорят, что коммутатор поддерживает *мгновенный неблокирующий режим.* Это означает, что он может принимать и обрабатывать кадры от всех своих портов на максимальной скорости протокола независимо от того, обеспечиваются ли условия устойчивого равновесия между входным и выходным трафиком. Правда, обработка некоторых кадров при этом может быть неполной — при занятости выходного порта кадр помещается в буфер коммутатора.

Для поддержки мгновенного неблокирующего режима коммутатор должен обладать большей собственной производительностью, а именно она должна быть равна суммарной производительности его портов: $C_k = \sum C_{pi}$.

Приведенные соотношения справедливы для портов с любыми скоростями, то есть портов стандартов Ethernet со скоростью 10 Мбит/с, Fast Ethernet, Gigabit Ethernet, 10G и 100G Ethernet.

Способы, которыми осуществляется способность коммутатора поддерживать неблокирующий режим, могут быть разными. Необходимым требованием является умение процессора порта обрабатывать потоки кадров с максимальной для физического уровня этого порта скоростью. Ранее было подсчитано, что максимальная производительность порта Ethernet стандарта 10 Мбит/с равна 14 880 кадров (минимальной длины) в секунду. Это означает, что процессоры портов Ethernet стандарта 10 Мбит/с неблокирующего коммутатора должны поддерживать продвижение кадров со скоростью 14 880 кадров в секунду. Как будет видно дальше, более скоростные версии Ethernet сохраняют формат кадра Ethernet и сокращают межкадровый интервал пропорционально увеличению битовой скорости версии (то есть в 10 раз при повышении скорости в 10 раз). Поэтому максимальные значения скорости продвижения кадров также растут пропорционально росту битовой скорости, например Fast Ethernet обеспечивает максимальную скорость продвижения в 148 800 кадров в секунду, а Gigabit Ethernet - в 1 488 000 кадров в секунду. Соответственно должна расти скорость продвижения коммутатора Ethernet с высокоскоростными портами.

Однако только адекватной производительности процессоров портов недостаточно для того, чтобы коммутатор был неблокирующим. Необходимо, чтобы достаточной производительностью обладали все элементы архитектуры коммутатора, включая центральный процессор, общую память, шины, соединяющие отдельные модули между собой, саму архитектура коммутатора. В принципе, задача создания неблокирующего коммутатора аналогична задаче создания высокопроизводительного компьютера – в обоих случаях она решается комплексно: за счет соответствующей архитектуры объединения модулей в едином устройстве и адекватной производительности каждого отдельного модуля устройства.

11.4.6 Борьба с перегрузками

Даже в том случае, когда коммутатор является неблокирующим, нет гарантии того, что он во всех случаях справится с потоком кадров, направляемых на его порты. Неблокирующие коммутаторы тоже могут испытывать перегрузки и терять кадры из-за переполнения внутренних буферов.

Причина перегрузок обычно кроется не в том, что коммутатору не хватает производительности для обслуживания потоков кадров, а в ограниченной

пропускной способности конкретного выходного порта, которая определяется параметрами протокола. Другими словами, какой бы производительностью коммутатор ни обладал, всегда найдется такое распределение потоков кадров, которое приведет к перегрузке коммутатора из-за ограниченной производительности выходного порта коммутатора.

Возникновение таких перегрузок является платой за отказ от применения алгоритма доступа к разделяемой среде, так как в дуплексном режиме работы портов теряется контроль над потоками кадров, направляемых конечными узлами в сеть. В полудуплексном режиме, свойственном технологиям с разделяемой средой, поток кадров регулируется самим методом доступа к разделяемой среде. При переходе на дуплексный режим узлу разрешается отправлять кадры в коммутатор всегда, когда это ему нужно, поэтому в данном режиме коммутаторы сети могут сталкиваться с перегрузками, не имея при этом никаких средств «притормаживания» потока кадров.

Таким образом, если входной трафик неравномерно распределяется между выходными портами, легко представить ситуацию, когда на какой-либо выходной порт коммутатора будет направляться трафик с суммарной средней интенсивностью большей, чем протокольный максимум. На рисунке 2.34 показана как раз такая ситуация, когда на порт 3 коммутатора Ethernet от портов 1, 2, 4 и 6 направляется поток кадров размером в 64 байт с суммарной интенсивностью в 22 100 кадров в секунду. Вспомним, что максимальная скорость в кадрах в секунду для сегмента Ethernet составляет 14 880. Естественно, что когда кадры поступают в буфер порта со скоростью 22 100 кадров в секунду, а уходят со скоростью 14 880 кадров в секунду, то внутренний буфер выходного порта начинает неуклонно заполняться необработанными кадрами.

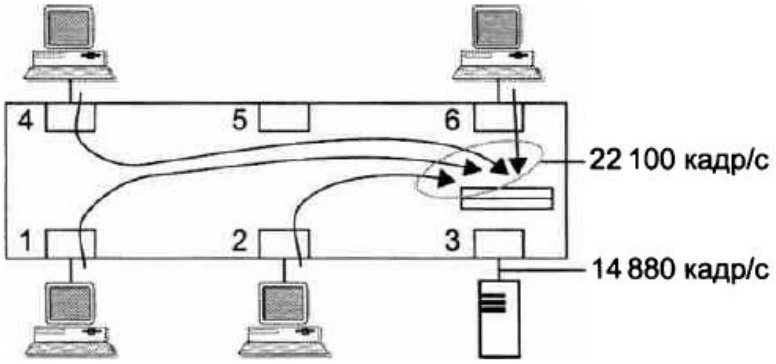


Рисунок 2.34 – Переполнение буфера порта
из-за несбалансированности трафика

В приведенном примере нетрудно подсчитать, что при размере буфера в 100 Кбайт полное заполнение буфера произойдет через 0,22 секунды после начала работы в таком интенсивном режиме. Увеличение размера буфера до 1 Мбайт даст увеличение времени заполнения буфера до 2,2 секунд, что также неприемлемо.

Проблему можно решить с помощью средств контроля перегрузки, которые были рассмотрены в главе 6.

Как мы знаем, существуют различные средства контроля перегрузки: управление очередями в коммутаторах, обратная связь, резервирование пропускной способности. На основе этих средств можно создать эффективную систему поддержки показателей QoS для трафика разных классов.

В этом разделе мы рассмотрим механизм обратной связи, который был стандартизован для сетей Ethernet в марте 1997 года как спецификация IEEE 802.3x. Механизм обратной связи 802.3x используется только в дуплексном режиме работы портов коммутатора. Этот механизм очень важен для коммутаторов локальных сетей, так как он позволяет сократить потери кадров из-за переполнения буферов.

Спецификация 802.3x вводит новый подуровень в стеке протоколов Ethernet — подуровень управления уровня MAC. Он располагается над уровнем MAC и является необязательным.

Кадры этого подуровня могут использоваться в различных целях, но пока в стандартах Ethernet для них определена только одна задача — приостановка передачи кадров другими узлами на определенное время.

Коммутатор использует кадр подуровня управления в том случае, когда ему нужно на время приостановить поступление кадров от соседнего узла, чтобы разгрузить свои внутренние очереди.

В качестве адреса назначения можно указывать зарезервированное для этой цели значение группового адреса 01-80-C2-00-00-01. Это удобно, когда соседний узел также является коммутатором (так как порты коммутатора не имеют уникальных MAC-адресов). Если сосед — конечный узел, можно также использовать уникальный MAC-адрес.

В поле кода операции подуровня управления указывается шестнадцатеричный код 00-

01, поскольку, как уже было отмечено, пока определена только одна операция подуровня управления — она называется PAUSE (пауза) и имеет шестнадцатеричный код 00-01.

В поле параметров подуровня управления указывается время, на которое узел, получивший такой код, должен прекратить передачу кадров узлу, отправившему кадр с операцией PAUSE. Время измеряется в 512 битовых интервалах конкретной реализации Ethernet, диапазон возможных вариантов приостановки равен 0-65 535.

Как видно из описания, этот механизм обратной связи относится к типу 2 в соответствии с классификацией, приведенной в главе 6. Специфика его состоит в том, что в нем предусмотрена только одна операция — приостановка на

определенное время. Обычно же в механизмах этого типа используются две операции — приостановка и возобновление передачи кадров.

Проблема, иллюстрируемая рис. 12.9, может быть решена и другим способом: применением так называемого магистрального, или восходящего (uplink), порта. Магистральные порты в коммутаторах Ethernet — это порты следующего уровня иерархии скорости по сравнению с портами, предназначенными для подключения пользователей. Например, если коммутатор имеет 12 портов Ethernet стандарта 10 Мбит/с, то магистральный порт должен быть портом Fast Ethernet, чтобы его скорость была достаточна для передачи до 10 потоков от входных портов. Обычно низкоскоростные порты коммутатора служат для соединения с пользовательскими компьютерами, а магистральные порты — для подключения либо сервера, к которому обращаются пользователи, либо коммутатора более высокого уровня иерархии.

На рис. 12.10 показан пример коммутатора, имеющего 24 порта стандарта Fast Ethernet со скоростью 100 Мбит/с, к которым подключены пользовательские компьютеры, и один порт стандарта Gigabit Ethernet со скоростью 1000 Мбит/с, к которому подключен сервер. При такой конфигурации коммутатора вероятность перегрузки портов существенно снижается по сравнению с вариантом, когда все порты поддерживают одинаковую скорость. Хотя возможность перегрузки по-прежнему существует, для этого необходимо, чтобы более 10 пользователей одновременно обменивались с сервером данными со средней скоростью, близкой к максимальной скорости их соединений, а такое событие достаточно маловероятно.

Из приведенного примера видно, что вероятность перегрузки портов коммутаторов зависит от распределения трафика между его портами, кроме того, понятно, что даже при хорошем соответствии скорости портов наиболее вероятному распределению трафика полностью исключить перегрузки невозможно.

Поэтому в общем случае для уменьшения потерь кадров из-за перегрузок нужно применять оба средства: подбор скорости портов для наиболее вероятного распределения трафика в сети и протокол 802.3x для снижения скорости источника трафика в тех случаях, когда перегрузки все-таки возникают.

11.5 Скоростные версии Ethernet

Скорость 10 Мбит/с первой стандартной версии Ethernet долгое время удовлетворяла потребности пользователей локальных сетей. Однако в начале 90-х годов начала ощущаться недостаточная пропускная способность Ethernet, так как скорость обмена с сетью стала существенно меньше скорости внутренней

шины компьютера. Кроме того, начали появляться новые мультимедийные приложения, гораздо более требовательные к скорости сети, чем их текстовые предшественники. В поисках решения проблемы ведущие производители сетевого оборудования начали интенсивные работы по повышению скорости Ethernet при сохранении главного достоинства этой технологии — простоты и невысокой стоимости оборудования.

Результатом стало появление новых скоростных стандартов Ethernet: Fast Ethernet (скорость 100 Мбит/с), Gigabit Ethernet (1000 Мбит/с, или 1 Гбит/с), 10G Ethernet (10 Гбит/с) и 100G Ethernet (100 Гбит/с).

Разработчикам новых скоростных стандартов Ethernet удалось сохранить основные черты классической технологии Ethernet, и прежде всего простой способ обмена кадрами без встраивания в технологию сложных контрольных процедур. Этот фактор оказался решающим в соревновании технологий локальных сетей, так как выбор пользователей всегда склонялся в пользу простого наращивания скорости сети, а не в пользу решений, связанных с более эффективным расходованием той же самой пропускной способности с помощью более сложной и дорогой технологии.

Значительный вклад в «победу» Ethernet внесли также коммутаторы локальных сетей, так как их успех привел к отказу от разделяемой среды, где технология Ethernet всегда была уязвимой из-за случайного характера метода доступа. Начиная с версии 10G Ethernet, разработчики перестали включать вариант работы на разделяемой среде в описание стандарта.

Повышение скорости работы Ethernet было достигнуто за счет улучшения качества кабелей, применяемых в компьютерных сетях, совершенствования методов кодирования данных при их передаче по витым парам (а также использования параллельных потоков данных, то есть за счет совершенствования физического уровня технологии).

11.5.1 Физические уровни технологии Fast Ethernet

Разработчикам технологии Fast Ethernet удалось обеспечить ее преемственность с классической технологией Ethernet 10 Мбит/с.

Поэтому все отличия технологий Fast Ethernet и Ethernet проявляются на физическом уровне (рисунок 12.35). Уровни MAC и LLC в Fast Ethernet остались абсолютно теми же, и их описывают прежние главы стандартов 802.3 и 802.2.

Рассматривая технологию Fast Ethernet, мы будем изучать только варианты ее физического уровня.

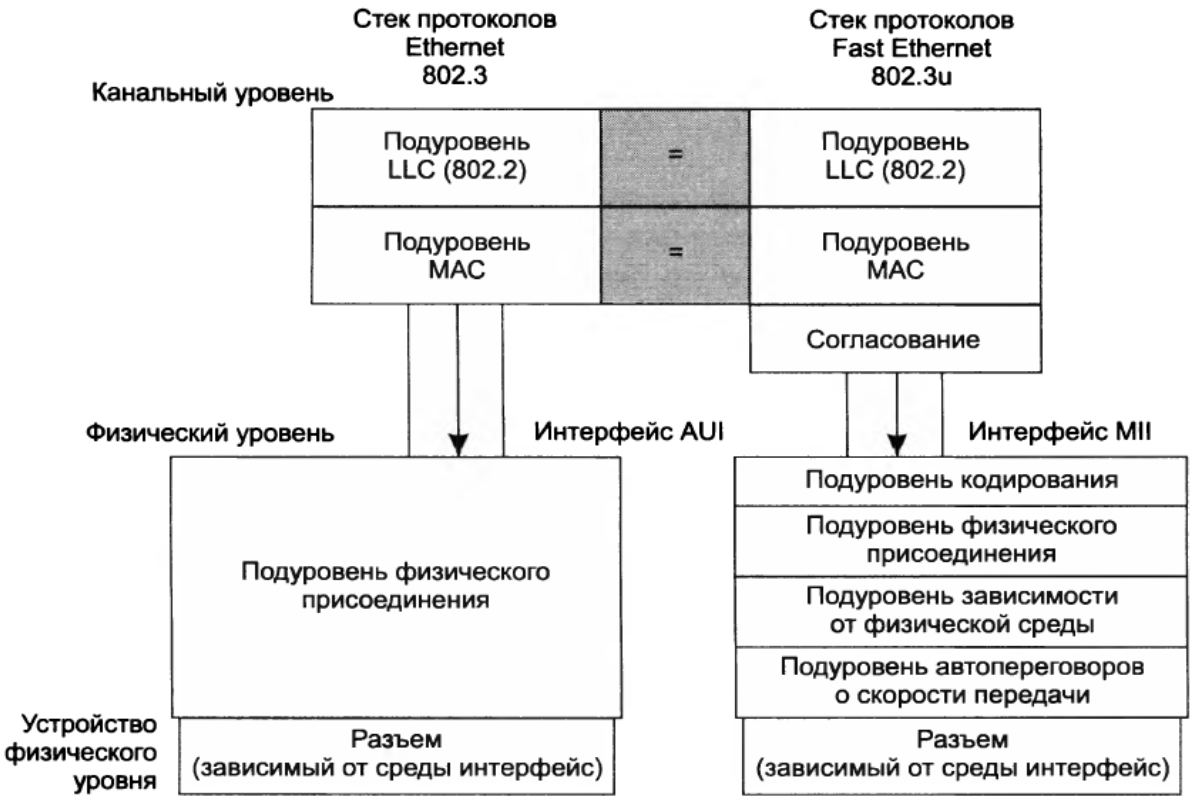


Рисунок 2.35 – Отличия технологий Fast Ethernet и Ethernet

Организация физического уровня технологии Fast Ethernet является модульной. Это объясняется тем, что технология Fast Ethernet изначально была рассчитана на применение различных типов физической среды и кодирования, модульность физического уровня позволяет достичь этой цели достаточно легко. Различные же варианты физической среды Ethernet 10 Мбит/с разрабатывались разными организациями в разное время, отсюда и отсутствие гибкости в построении физического уровня. Нужно подчеркнуть, что этот модульный подход был впоследствии применен и во всех других более скоростных вариантах Ethernet, включая 100G Ethernet.

Физический уровень Fast Ethernet состоит из трех модулей:

- *независимый от среды интерфейс (Media Independent Interface, MII).* Этот интерфейс поддерживает независимый от физической среды способ обмена данными между подуровнями MAC и PHY;

- *модуль согласования (Reconciliation)* нужен для того, чтобы уровень MAC, рассчитанный ранее на интерфейс AUI, мог работать с физическим уровнем через интерфейс MII;

- *устройство физического уровня (Physical Layer Device, PHY)* состоит, в свою очередь, из нескольких подуровней (см. рисунок 2.35):

- подуровня кодирования данных (Physical Coding Sublayer, PCS), преобразующего поступающие от уровня MAC байты в символы логического кода, например, 4B/5B;

- подуровней физического присоединения (Physical Media Attachment, PMA) и зависимости от физической среды (Physical Media Dependent, PMD), которые обеспечивают формирование сигналов в соответствии с методом физического кодирования, например, NRZI;

- подуровня автопереговоров, который позволяет двум взаимодействующим портам автоматически выбрать наиболее эффективный режим работы, например, полудуплексный или дуплексный (этот подуровень является факультативным).

Fast Ethernet поддерживает три варианта физической среды:

- *волоконно-оптический многомодовый кабель (два волокна);*
- *витая пара категории 5 (две пары);*
- *витая пара категории 3 (четыре пары).*

Коаксиальный кабель, давший миру первую сеть Ethernet, в число разрешенных сред передачи данных технологии Fast Ethernet не попал. Это общая тенденция многих новых технологий, поскольку на небольших расстояниях витая пара категории 5 позволяет передавать данные с той же скоростью, что и коаксиальный кабель, но сеть получается более дешевой и удобной в эксплуатации. На больших расстояниях оптическое волокно обладает гораздо более широкой полосой пропускания, чем коаксиал, а стоимость сети получается ненамного выше, особенно если учесть высокие затраты на поиск и устранение неисправностей в крупной кабельной коаксиальной системе.

Официальный стандарт 802.3 установил три различные спецификации для физического уровня Fast Ethernet и дал им следующие названия:

- *100Base-TX* — для двухпарного кабеля на неэкранированной витой паре UTP категории 5;

- *100Base-T4* — для четырехпарного кабеля на неэкранированной витой паре UTP категории 3,4 или 5;

- *100Base-FX* — для многомодового оптоволоконного кабеля с двумя волокнами.

Для всех трех стандартов справедливы перечисленные далее утверждения и характеристики.

Форматы кадров технологии Fast Ethernet не отличаются от форматов кадров технологий Ethernet 10 Мбит/с.

Межкадровый интервал равен 0,96 мкс, а битовый интервал - 10 нс. Все временные параметры алгоритма доступа (интервал отсрочки, время передачи кадра минимальной длины и т. гг.), измеренные в битовых интервалах, остались прежними.

Признаком свободного состояния среды является передача по ней символа простоя источника — соответствующего избыточного кода (а не отсутствие сигналов, как в стандартах Ethernet 10 Мбит/с).

Версия 100Base-T4 носила промежуточный характер, так как она позволяла повысить скорость классического варианта Ethernet в 10 раз, не меняя кабельную систему здания. Так как большинство предприятий и организаций достаточно быстро заменили кабели категории 3 кабелями категории 5, то необходимость в версии 100Base-T4 отпала и оборудование с таким портами перестало выпускаться. Поэтому далее мы рассмотрим детали только спецификаций 100Base-FX и 100Base-TX.

Спецификация 100Base-FX определяет работу протокола Fast Ethernet по многомодовому оптоволокну в полудуплексном и дуплексном режимах. Подуровень кодирования 100BaseFX преобразует байты, получаемые от уровня MAC, в избыточные коды 4B/5B (мы рассматривали этот код в главе 8). В качестве физического метода кодирования используется код IMDD («свет-темнота»).

Избыточный код 4B/5B ко времени разработки технологии Fast Ethernet уже показал свою эффективность в сетях FDDI, поэтому он без изменений был перенесен в спецификацию 100Base-FX/TX. Напомним, что в этом методе каждые четыре бита данных подуровня MAC (называемые символами) представляются пятью битами. Избыточный бит позволяет применить потенциальные коды при представлении каждого из пяти битов в виде электрических или оптических импульсов.

Существование запрещенных комбинаций символов позволяет отбраковывать ошибочные символы, что повышает устойчивость работы сетей 100Base-FX/TX. Так, в Fast Ethernet признаком того, что среда свободна, стала повторяющаяся передача одного из запрещенных для кодирования пользовательских данных символа, а именно символа простоя источника *Idle* (11111). Такой способ позволяет приемнику всегда находиться в синхронизме с передатчиком.

Для отделения кадра Ethernet от символов простоя источника используется комбинация символов начального ограничителя кадра — пара символов *J* (11000) и *K* (10001) кода 4B/5B.

После преобразования 4-битных порций кодов MAC в 5-битные порции физического уровня их необходимо представить в виде оптических или электрических сигналов в кабеле, соединяющем узлы сети.

В спецификации 100Base-TX в качестве среды передачи данных используется витая пара UTP категории 5. Как и в спецификации 100Base-FX, здесь применяется избыточный код 4B/5B, а для физического кодирования — код MLT-3 с тремя состояниями электрического сигнала (один из вариантов кода NRZ).

Основным отличием от спецификации 100Base-FX является наличие схемы автопереговоров для выбора режима работы порта.

Схема автопереговоров позволяет двум физически соединенным устройствам, которые поддерживают несколько стандартов физического уровня, отличающихся битовой скоростью и количеством витых пар, согласовать наиболее выгодный режим работы. Обычно процедура автопереговоров происходит при подсоединении сетевого адаптера, который может работать на скоростях 10 и 100 Мбит/с, к концентратору или коммутатору.

Всего в настоящее время определено пять различных режимов работы, которые могут поддерживать устройства 100Base-TX/T4 на витых парах:

- 10Base-T;
- дуплексный режим 10Base-T;
- 100Base-TX;
- 100Base-T4;
- дуплексный режим 100Base-TX.

Режим 10Base-T имеет самый низкий приоритет в переговорном процессе, а дуплексный режим 100Base-TX — самый высокий.

Переговорный процесс происходит при включении питания устройства, а также может быть инициирован в любой момент модулем управления устройства. Устройство, начавшее процесс автопереговоров, посылает своему партнеру пачку специальных импульсов FLP (Fast Link Pulse), в которой содержится 8-битное слово, кодирующее предлагаемый режим взаимодействия, начиная с самого приоритетного, поддерживаемого данным узлом. Импульсы FLP имеют длительность 100 не, как и импульсы LIT, используемые для тестирования целостности физического соединения в стандарте 10Base-T, однако вместо передачи одного импульса LIT через каждые 16 мс здесь через тот же интервал передается пачка импульсов FLP.

Если узел-партнер имеет функцию автопереговоров и также способен поддерживать предложенный режим, он отвечает пачкой импульсов FLP, в которой подтверждает этот режим, и на этом переговоры заканчиваются. Если же узел-партнер не может поддерживать запрошенный режим, то он указывает в своем ответе имеющийся в его распоряжении следующий по степени приоритетности режим и этот режим выбирается в качестве рабочего.

Характеристики производительности Fast Ethernet определяются аналогично характеристикам версии Ethernet 10 Мбит/с с учетом неизменного формата кадра, умножения на 10 битовой скорости (в 10 раз больше) и межкадрового интервала (в 10 раз меньше).

11.5.2 Gigabit Ethernet

Достаточно быстро после появления на рынке продуктов Fast Ethernet сетевые интеграторы и администраторы при построении корпоративных сетей почувствовали определенные ограничения. Во многих случаях серверы, подключенные по 100-мегабитному каналу, перегружали магистрали сетей, также работающие на скорости 100 Мбит/с, - магистрали FDDI и Fast Ethernet. Ощущалась потребность в следующем уровне иерархии скоростей. Стандарт Ethernet с битовой скоростью 1000 Мбит/с, получивший название Gigabit Ethernet, был принят в 1998 году.

Проблемы совместимости. Основная идея разработчиков стандарта Gigabit Ethernet состояла в максимальном сохранении идей классической технологии Ethernet при достижении битовой скорости в 1000 Мбит/с.

В результате дебатов были приняты следующие решения:

- сохраняются все форматы кадров Ethernet;
- по-прежнему существует полудуплексная версия протокола, поддерживающая метод доступа CSMA/CD;
- поддерживаются все основные виды кабелей, используемых в Ethernet и Fast Ethernet, в том числе волоконно-оптический кабель, витая пара категории 5, экранированная витая пара.

Несмотря на то что в Gigabit Ethernet не стали встраивать новые функции, поддержание даже достаточно простых функций классического стандарта Ethernet на скорости 1 Гбит/с потребовало решения нескольких сложных задач.

Обеспечение приемлемого диаметра сети для работы на разделяемой среде. В связи с ограничениями, накладываемыми методом CSMA/CD на длину кабеля, версия Gigabit Ethernet для разделяемой среды допускала бы длину сегмента всего в 25 м при сохранении размера кадров и всех параметров метода CSMA/CD неизменными. Так как существует большое количество применений, требующих диаметра сети 100 м, необходимо было каким-то образом решить эту задачу за счет минимальных изменений в технологии Fast Ethernet.

Достижение битовой скорости 1000 Мбит/с на оптическом кабеле. Технология Fibre Channel, физический уровень которой был взят за основу оптоволоконной версии Gigabit Ethernet, обеспечивала скорость передачи данных всего в 800 Мбит/с.

Использование в качестве кабеля витой пары. Такая задача на первый взгляд кажется неразрешимой — ведь даже для 100-мегабитных протоколов требуются достаточно сложные методы кодирования, чтобы уложить спектр сигнала в полосу пропускания кабеля.

Для решения этих задач разработчикам технологии Gigabit Ethernet пришлось внести изменения не только в физический уровень, как это было в случае Fast Ethernet, но и в уровень MAC.

Средства обеспечения диаметра сети в 200 м на разделяемой среде. Для расширения максимального диаметра сети Gigabit Ethernet до 200 м в полудуплексном режиме разработчики технологии предприняли достаточно естественные меры, в основе которых лежало известное соотношение времени передачи кадра минимальной длины и времени оборота (PDV).

Минимальный размер кадра был увеличен (без учета преамбулы) с 64 до 512 байт, или до 4096 бит. Соответственно время оборота увеличилось до 4095 битовых интервалов, что при использовании одного повторителя сделало допустимым диаметр сети около 200 м.

Для увеличения длины кадра до величины, требуемой в новой технологии, сетевой адаптер должен дополнить поле данных до длины 448 байт так называемым расширением, представляющим собой поле, заполненное нулями. Формально минимальный размер кадра не изменился, он по-прежнему равняется 64 байт, или 512 бит, и объясняется это тем, что поле расширения помещается после поля контрольной суммы кадра (FCS). Соответственно значение этого поля не включается в контрольную сумму и не учитывается при указании длины поля данных в поле длины. Поле расширения является просто расширением сигнала несущей частоты, необходимым для корректного обнаружения коллизий.

Для сокращения накладных расходов в случае использования слишком длинных кадров при передаче коротких квитанций разработчики стандарта разрешили конечным узлам передавать несколько кадров подряд без возвращения среды другим станциям. Такой режим получил название режима пульсаций. Станция может передать подряд несколько кадров с общей длиной не более 65 536 бит, или 8192 байт. При передаче нескольких небольших кадров станции можно не дополнять первый кадр до размера в 512 байт за счет поля расширения, а передавать несколько кадров подряд до исчерпания предела в 8192 байт (в этот предел входят все байты кадра, в том числе преамбула, заголовок, данные и контрольная сумма). Предел 8192 байт называется длиной пульсации. Если предел длины пульсации достигается в середине кадра, то кадр разрешается передать до конца. Увеличение «совмещенного» кадра до 8192 байт несколько задерживает доступ к разделяемой среде других станций, но при скорости 1000 Мбит/с эта задержка не столь существенна.

Спецификации физической среды стандарта Gigabit Ethernet. Для поддержания различных физических сред физический уровень Gigabit Ethernet имеет такую же модульную структуру, как и физический уровень Fast Ethernet, с

тем отличием, что вместо интерфейса МИ в нем применяется интерфейс GMII (Gigabit МП), работающий на скорости 1 Гбит/с.

В стандарте 802.3z определены следующие типы физической среды:

- одномодовый волоконно-оптический кабель;
- многомодовый волоконно-оптический кабель 62,5/125;
- многомодовый волоконно-оптический кабель 50/125;
- экранированный сбалансированный медный кабель.

Для передачи данных по многомодовому волоконно-оптическому кабелю стандарт предписывает применение излучателей, работающих на двух длинах волн: 850 и 1300 нм. Применение светодиодов с длиной волны 850 нм объясняется тем, что они намного дешевле, чем светодиоды, работающие на волне 1300 нм, хотя при этом максимальная длина кабеля уменьшается, так как затухание многомодового оптоволокна на волне 850 нм более чем в два раза выше, чем на волне 1300 нм.

Для многомодового оптоволокна стандарт Gigabit Ethernet определяет спецификации 1000Base-SX и 1000Base-LX. В первом случае используется длина волны 850 нм (S означает Short Wavelength - короткая длина волны), а во втором - 1300 нм (L означает Long Wavelength - длинная длина волны). Спецификация 1000Base-SX разрешает использовать только многомодовый кабель, при этом его максимальная длина составляет около 500 м.

Для спецификации 1000Base-LX в качестве источника излучения всегда применяется полупроводниковый лазерный диод с длиной волны 1300 нм. Спецификация 1000Base-LX позволяет работать как с многомодовым (максимальное расстояние до 500 м), так и с одномодовым кабелем (максимальное расстояние зависит от мощности передатчика и качества кабеля и может доходить до нескольких десятков километров).

В спецификациях 1000Base-SX и 1000Base-LX подуровень кодирования преобразует байты уровня MAC в коды 8B/10B (а не 4B/5B, как в стандарте Fast Ethernet).

Gigabit Ethernet на витой паре категории 5. Как известно, каждая пара кабеля категории 5 имеет гарантированную полосу пропускания до 100 МГц. Организовать передачу по такому кабелю данных со скоростью 1000 Мбит/с очень трудно, так как применяемые в локальных сетях методы кодирования имеют на такой тактовой частоте спектр с шириной, намного превышающий 100 МГц. В качестве решения было предложено организовать параллельную передачу данных одновременно по всем четырем парам кабеля.

Это сразу снизило скорость передачи данных по каждой паре до 250 Мбит/с. Однако и для такой скорости необходимо было придумать метод кодирования со спектром, не превышающим 100 МГц. Усложняло задачу и то обстоятельство, что

стандарт Gigabit Ethernet должен поддерживать не только полудуплексный, но и дуплексный режим. На первый взгляд кажется, что одновременное использование четырех пар лишает сеть возможности работы в дуплексном режиме, так как не остается свободных пар для одновременной передачи данных в двух направлениях - от узла и к узлу.

Тем не менее проблемная группа 802.3ab нашла решения обеих проблем.

Для физического кодирования данных был применен код PAM5 с пятью уровнями потенциала: -2, -1, 0, +1, +2. В этом случае за один такт по одной паре передается 2,322 бит информации ($\log_2 5$). Следовательно, для достижения скорости 250 Мбит/с тактовую частоту 250 МГц можно уменьшить в 2,322 раза. Разработчики стандарта решили использовать несколько более высокую частоту, а именно 125 МГц. При этой тактовой частоте код PAM5 имеет спектр уже, чем 100 МГц, то есть он может быть передан без искажений по кабелю категории 5.

В каждом такте передается не 2,322 х 4 = 9,288 бит информации, а 8. Это и дает искомую суммарную скорость 1000 Мбит/с. Передача ровно восьми битов в каждом такте достигается за счет того, что подуровень кодирования PCS преобразует байты, получаемые от уровня MAC через интерфейс GMII, в логический код 4D-PAM5, который состоит из четырех символов, каждый их которых имеет пять состояний (пять состояний символа соответствует пяти состояниям физического кода PAM5).

При кодировании информации используются не все 625 ($5^4 = 625$) комбинаций кода PAM5, а только 256 ($2^8 = 256$). Оставшиеся комбинации приемник задействует для контроля принимаемой информации и выделения правильных комбинаций на фоне шума.

Для организации дуплексного режима разработчики спецификации 802.3ab применили технику выделения принимаемого сигнала из суммарного. Два передатчика работают навстречу друг другу по каждой из четырех пар в одном и том же диапазоне частот.

Для отделения принимаемого сигнала от собственного приемник вычитает из результирующего сигнала известный ему свой сигнал. Естественно, что это не простая операция и для ее выполнения используются специальные процессоры цифровой обработки сигнала (Digital Signal Processor, DSP).

Вариант технологии Gigabit Ethernet на витой паре расширил процедуру автопереговоров, введенную стандартом 100Base-T, за счет включения туда дуплексного и полудуплексного режимов работы на скорости 1000 Мбит/с. Поэтому порты многих коммутаторов Ethernet на витой паре являются универсальными в том смысле, что могут работать на любой из трех скоростей (10, 100 или 1000 Мбит/с).

Характеристики производительности Gigabit Ethernet зависят от того, использует ли коммутатор режим передачи кадров с расширением или же передает их в режиме пульсаций.

В режиме пульсаций на периоде пульсации мы получаем характеристики, в 10 раз отличающиеся от характеристик Fast Etherhet

11.5.3 10G Ethernet

Стандарт 10G Ethernet определяет только дуплексный режим работы, поэтому он используется исключительно в коммутируемых локальных сетях. Формально этот стандарт имеет обозначение IEEE 802.3ae и является дополнением к основному тексту стандарта 802.3. Формат кадра остался неизменным, при этом расширение кадра, введенное в стандарте Gigabit Ethernet, не используется, так как нет необходимости обеспечивать распознавание коллизий.

Стандарт 802.3ae, принятый в 2002 году, описывает несколько новых спецификаций физического уровня, которые взаимодействуют с уровнем MAC через новый интерфейс XGMII (extended Gigabit Medium Independent Interface — расширенный интерфейс независимого доступа к гигабитной среде). Интерфейс XGMII предусматривает параллельный обмен четырьмя байтами, образующими четыре потока данных. Как мы видим, идея распараллеливания потоков данных, хорошо зарекомендовавшая себя в предыдущих версиях Ethernet как средство снижения требований к полосе пропускания отдельного канала, здесь нашла отражение в структуре интерфейса между уровнем MAC и физическим уровнем. Наличие четырех независимых потоков на входе физического уровня упрощает организацию параллельных потоков данных в приемопередатчиках Ethernet.

Стандарт 10G Ethernet определяет три группы физических интерфейсов:

- *10GBase-X;*
- *10Gbase-R4;*
- *10GBase-W.*

Они отличаются способом логического кодирования данных: в варианте 10Base-X применяется код 8B/10B, в остальных двух — код 64B/66B. Все они для передачи данных используют оптическую среду.

Группа 10GBase-X в настоящее время состоит из одного интерфейса подуровня PMD — 10GBase-LX4. Буква L говорит о том, что информация передается с помощью волн второго диапазона прозрачности, то есть 1310 нм. Информация в каждом направлении передается одновременно с помощью четырех волн (что отражает цифра 4 в названии интерфейса), которые мультиплексируются на основе техники WDM. Каждый из четырех потоков интерфейса XGMII передается в оптическом волокне со скоростью 2,5 Гбит/с.

Максимальное расстояние между передатчиком и приемником стандарта 10GBase-LX4 на многомодовом волокне равно 200-300 м (в зависимости от полосы пропускания волокна), на одномодовом — 10 км.

В каждой из групп 10GBase-R и 10GBase-W может быть три варианта подуровня PMD: S, L и E в зависимости от используемого для передачи информации диапазона волн — 850, 1310 или 1550 нм соответственно. Таким образом, существуют интерфейсы 10GBase-SR, 10GBase-LR и 10GBase-ER, а также 10GBase-SW, 10GBase-LW, 10GBase-EW. Каждый из них передает информацию с помощью одной волны соответствующего диапазона.

Спецификации 10GBase-R обеспечивают эффективную скорость передачи данных в 10 Гбит/с, для этого битовая скорость оборудования равна 10,3125 Гбит/с (увеличение битовой скорости необходимо для компенсации избыточности кода 64B/66B).

В отличие от 10GBase-R физические интерфейсы группы 10GBase-W обеспечивают скорость передачи и формат данных, совместимые с интерфейсом OTN ODU-2. Пропускная способность интерфейсов группы W равна 9,95328 Гбит/с, а эффективная скорость передачи данных — 9,58464 Гбит/с (часть пропускной способности тратится на заголовки кадров OTN). Из-за того, что скорость передачи информации у этой группы интерфейсов ниже, чем 10 Гбит/с, они могут взаимодействовать только между собой, то есть соединение, например, интерфейсов 10GBase-LR и 10Base-LW невозможно.

Физические интерфейсы, работающие в окне прозрачности E, обеспечивают передачу данных на расстояния до 40 км. Это позволяет строить не только локальные сети, но и сети мегаполисов, что нашло отражение в поправках к исходному тексту стандарта 802.3.

В 2006 году была принята спецификация 10GBase-T, которая дает возможность использовать знакомые администраторам локальных сетей кабели на витой паре. Правда, обязательным требованием является применение кабелей категории 6 или 6a: в первом случае максимальная длина кабеля не должна превышать 55 м, во втором — 100 м.

Стандарт 10G Ethernet развивается за счет пополнения его семейства физических интерфейсов новыми спецификациями, например, спецификацией 10GBase-KX4, предназначенной для работы по четырем проводникам шасси сетевых устройств.

11.5.4 100G и 40G Ethernet

Хотя 10 Гбит/с и является довольно высокой скоростью передачи данных, достаточной для многих приложений, например, для телевидения высокого разрешения, но и она по прошествии нескольких лет перестала удовлетворять потребности постоянно растущего трафика Интернета и новых приложений.

Поэтому в 2006 году рабочая группа IEEE 802.3 образовала группу по изучению высокоскоростного варианта Ethernet (High Speed Study Group, HSSG). Анализ ситуации показал, что целесообразно стандартизировать две новые скорости Ethernet - 40 и 100 Гбит/с. Первая скорость предназначалась для серверов, вторая - для интерфейсов коммутаторов и маршрутизаторов, работающих на магистралях сетей и агрегирующих потоки данных многих приложений. В результате в 2008 году была создана целевая группа 802.3ba, которая и предложила соответствующий вариант стандарта Ethernet, впервые описывающего упомянутые две скорости передачи данных. Этот стандарт вошел в общий стандарт 802.3-2012 как одна из частей (наряду с частями, описывающими остальные скорости Ethernet).

Архитектура стандарта 802.3ba обобщает подход, использованный в технологии 10G Ethernet, а именно распараллеливание общего потока данных от уровня MAC на несколько потоков. Этот подход позволяет снизить битовую скорость каждого из параллельных потоков, тем самым упрощая реализацию высокоскоростного приемопередатчика.

В стандарте 802.3ba распараллеливание применяется на двух этапах передачи данных от уровня MAC к уровню физического интерфейса (Physical Media Dependence, PMD). Сначала уровень согласования распараллеливает общий последовательный поток данных, поступающий от уровня MAC, на восемь потоков, которые параллельно поступают на подуровень физического кодирования PCS через интерфейс XLGMII (для скорости 40 Гбит/с, буквы XL и означают римское число 40) или интерфейс CGMII (для скорости 100 Гбит/с, от C - римское число 100).

Подуровень PCS выполняет кодирование данных, поступающих по восьми потокам, в соответствии с кодировкой 64B/66B (она одна для всех вариантов физического интерфейса), а затем направляет их четырьмя (для скорости 40 Гбит/с) или двадцатью (для скорости 100 Гбит/с) потоками на подуровни PMA и PMD, которые реализуются, как правило, отдельным модулем – приемопередатчиком (трансивером). В подуровнях PMA/PMD эти потоки могут группироваться в один или несколько каналов, передаваемых отдельными волнами (если применяется мультиплексирование WDM) или отдельными медными проводниками.

Выбор 20 потоков не случаен, он обеспечивает высокую гибкость для спецификаций физической среды, так как дает возможность сформировать 1, 2,4, 5,10 или 20 независимых физических каналов.

Рисунок 2.36 иллюстрирует работу подуровня PCS по распараллеливанию данных по потокам.

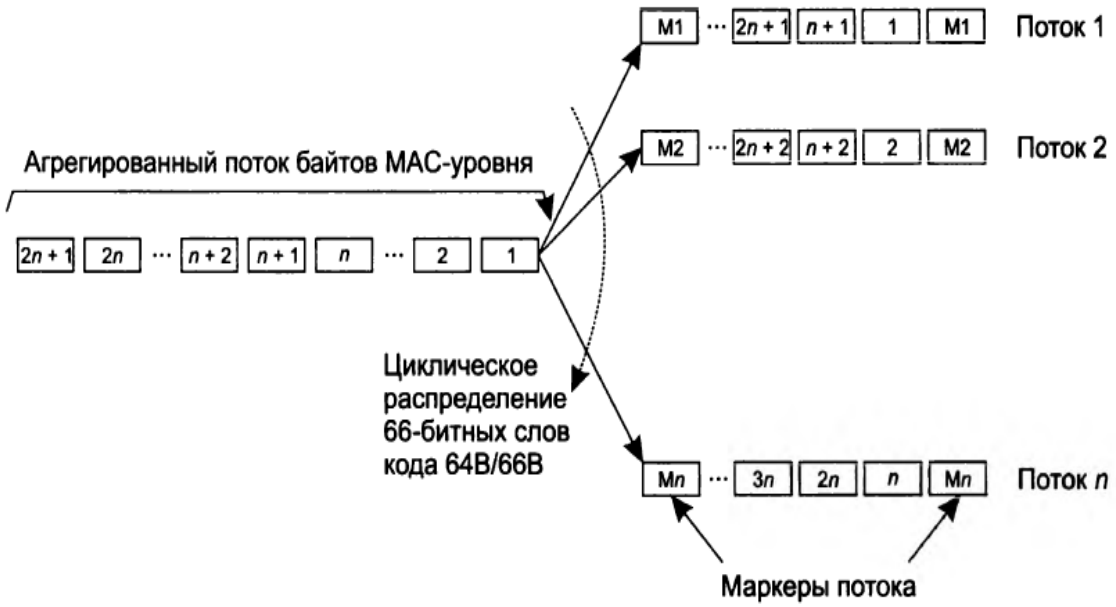


Рисунок 2.36 – Распределение данных подуровнем PCS по потокам

66-битные слова кода 64B/66B циклически распределяются между потоками данных. После каждых 16 384 слов в поток помещается специальный код маркера потока, который помогает подуровню PCS приемника правильно демультиплексировать потоки в общий поток. Для сохранения синхронности потока для вставки кода маркера из потока периодически удаляются коды межкадрового интервала (IPG).

Для 100 Gigabit Ethernet рабочей группой IEEE 802.3 разработаны различные стандарты физического уровня (таблица 2.6). По назначению они делятся на стандарты, предназначенные для работы в пределах шасси одного устройства (стандарты KR), для соединения устройств в пределах одной или нескольких стоек (CR и T), одного здания (SR и FR) или же для создания глобальных соединений между различными центрами данных (LR и ER). Почти все варианты физического уровня 100GE обеспечивают необходимую суммарную битовую скорость за счет использования нескольких параллельных потоков данных — как видно из таблицы, четырех или десяти. Эти параллельные потоки образуются либо отдельными проводниками (печатными проводниками для вариантов KR, витыми парами для варианта T, парами твинаксиального кабеля для вариантов CR или же парами оптических волокон для варианта SR), либо отдельными волнами технологии WDM в вариантах LR и ER.

Таблица 2.7 – Стандарты физического уровня для технологии 100 Gigabit Ethernet

Гарантированное расстояние и тип среды	40 Gigabit Ethernet	100 Gigabit Ethernet
>1 м шасси	40GBase-KR4	
>7 м твинаксиальный медный кабель	40GBase-CR4	100GBase-CR10
>100 м OM3* MMF	40GBase-SR4	100GBase-SR10
>150 м OM4 MMF	40GBase-SR4	100GBase-SR10
>10 км SMF	40GBase-LR4	100GBase-LR10
>40 км SMF		100GBase-ER10
>2 км SMF	40GBase-FR	
>30 м витой пары категории 8 (4 пары)	40GBase-T*	

11.6 Беспроводные локальные сети IEEE 802.11

11.6.1 Проблемы и области применения беспроводных локальных сетей

Беспроводные локальные сети (Wireless Local Area Network, WLAN) в некоторых случаях являются предпочтительным по сравнению с проводными сетями решением, а иногда просто единственно возможным. В WLAN сигнал распространяется с помощью электромагнитных волн высокой частоты. Современные беспроводные локальные сети позволяют передавать данные на скоростях до нескольких гигабит в секунду.

Преимущество беспроводных локальных сетей очевидно - их проще и дешевле разворачивать и модифицировать, так как вся громоздкая кабельная инфраструктура оказывается излишней. Еще одно преимущество - обеспечение мобильности пользователей. Сегодня беспроводные локальные технологии успешно применяются во многих типах сетей: в домашних сетях, в сетях аэропортов, вокзалов, кафе и других публичных местах, для организации временных сетей на различных конференциях и совещаниях, в сетях исторических зданий с уникальной архитектурой, исключающей возможность прокладки кабелей, а также как городские сети, предоставляющие доступ в Интернет на всей территории города.

Однако за эти преимущества беспроводные сети расплачиваются длинным перечнем проблем, которые несет с собой неустойчивая и непредсказуемая беспроводная среда.

Помехи от разнообразных бытовых приборов и других телекоммуникационных систем, атмосферные помехи и отражения сигнала создают серьезные трудности для надежного приема информации. Локальные сети – это прежде всего сети зданий, а распространение радиосигнала внутри

здания еще сложнее, чем вне его. В стандарте IEEE 802.11 приводится изображение распределения интенсивности сигнала (рисунок 2.37).

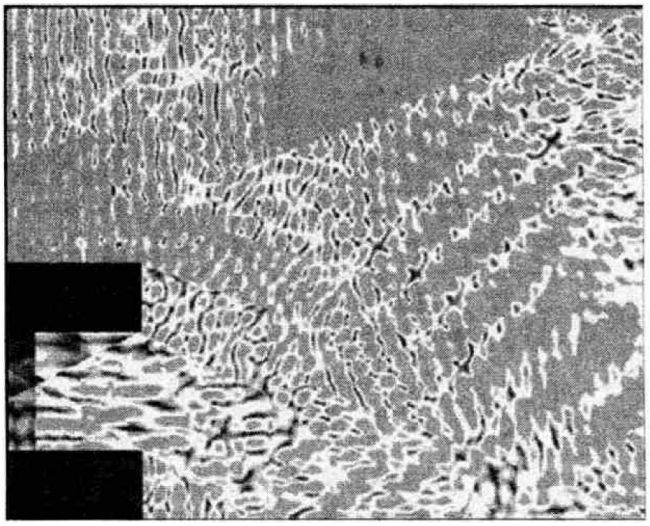


Рисунок 2.37 – Распределение интенсивности радиосигнала

В стандарте подчеркивается, что это – статическое изображение, в действительности картина является динамической, и при перемещении объектов в комнате распределение сигнала может существенно измениться.

Методы расширения спектра помогают снизить влияние помех на полезный сигнал, кроме того, в беспроводных сетях широко используются прямая коррекция ошибок (FEC) и протоколы с повторной передачей потерянных кадров.

Неравномерное распределение интенсивности сигнала приводит не только к битовым ошибкам передаваемой информации, но и к неопределенности зоны покрытия беспроводной локальной сети. В проводных локальных сетях такой проблемы нет, те и только те устройства, которые подключены к кабельной системе здания или кампуса, получают сигналы и участвуют в работе LAN. Беспроводная локальная сеть не имеет точной области покрытия. Часто используемое изображение такой области в форме шестиугольника или круга является не чем иным, как абстракцией. В действительности сигнал может быть настолько ослаблен, что устройства, находящиеся в предполагаемых пределах зоны покрытия, вообще не могут принимать и передавать информацию.

Рисунок 2.37 хорошо иллюстрирует такую ситуацию. Подчеркнем, что с течением времени ситуация с распределением сигнала может измениться вместе с изменением состава LAN. По этой причине даже технологии, рассчитанные на фиксированные (не мобильные) узлы сети, должны учитывать то, что беспроводная локальная сеть является неполносвязной. Даже если считать, что сигнал распространяется идеально во все стороны, образованию полносвязной топологии может мешать то, что радиосигнал затухает пропорционально квадрату расстояния от источника. Поэтому при отсутствии базовой станции некоторые

пары узлов не смогут взаимодействовать из-за того, что расположены за пределами зоны покрытия передатчиков партнера.

В примере на рисунке 2.38, а показана такая фрагментированная локальная сеть. Неполно- связность беспроводной сети порождает проблему доступа к разделяемой среде, известную под названием скрытого терминала. Проблема возникает в том случае, когда два узла находятся вне зон досягаемости друг друга (узлы А и С на рисунке 2.38, а), но существует третий узел В, который принимает сигналы как от А, так и от С. Предположим, что в радиосети используется традиционный метод доступа, основанный на прослушивании несущей, например, CSMA/CD. В данном случае коллизии будут возникать значительно чаще, чем в проводных сетях. Пусть, например, узел В занят обменом с узлом А. Узлу С сложно определить, что среда занята, он может посчитать ее свободной и начать передавать свой кадр. В результате сигналы в районе узла В исказятся, то есть произойдет коллизия, вероятность возникновения которой в проводной сети была бы неизмеримо ниже.

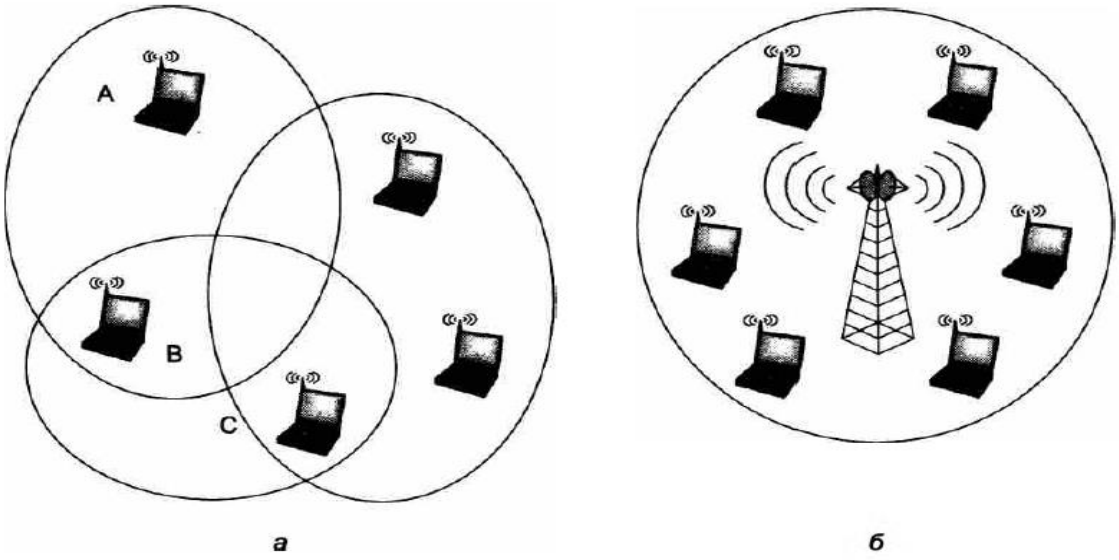


Рисунок 2.38 – Связность беспроводной локальной сети:
а – специализированная беспроводная сеть;
б – беспроводная сеть с базовой станцией

Распознавание коллизий затруднено в радиосети еще и потому, что сигнал собственного передатчика существенно подавляет сигнал удаленного передатчика и распознать искажение сигнала чаще всего невозможно.

В методах доступа, применяемых в беспроводных сетях, отказываются не только от прослушивания несущей, но и от распознавания коллизий.

Вместо этого в них используют методы предотвращения коллизий, включая методы опроса.

Применение базовой станции может улучшить связность сети (рисунок 2.38,6). Базовая станция обычно обладает большей мощностью, а ее антенна устанавливается так, чтобы более равномерно и беспрепятственно покрывать нужную территорию. В результате все узлы беспроводной локальной сети получают возможность обмениваться данными с базовой станцией, которая транзитом передает данные между узлами.

Далее будет рассмотрен самый популярный стандарт беспроводных локальных сетей – IEEE 802.11. Сети и оборудование IEEE.802.il также известны под названием Wi-Fi – по имени консорциума Wi-Fi1 Alliance, который занимается вопросам совместимости и сертификации оборудования стандартов IEEE 802.11.

11.6.2 Стандарт 802.11

Стандарт 802.11 поддерживает два типа топологий локальных сетей: с базовым и с расширенным наборами услуг.

Сеть с базовым набором услуг (Basic Service Set, BSS) образуется отдельными станциями, базовая станция отсутствует, узлы взаимодействуют друг с другом непосредственно (рисунок 2.39). Для того чтобы войти в сеть BSS, станция должна выполнить процедуру присоединения.

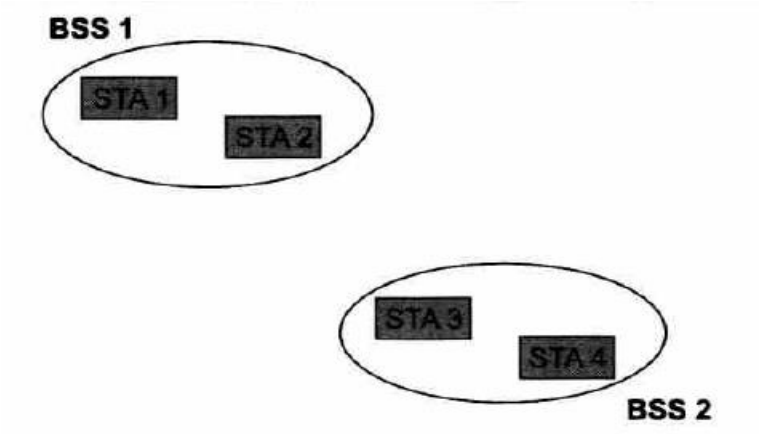


Рисунок 2.39 – Сети с базовым набором услуг

Сети BSS не являются традиционными сотами в отношении зон покрытия, они могут находиться друг от друга на значительном расстоянии, а могут частично или полностью перекрываться — стандарт 802.11 оставляет здесь свободу для проектировщика сети.

Станции могут использовать разделяемую среду для того, чтобы передавать данные:

- непосредственно друг другу в пределах одной сети BSS;
- в пределах одной сети BSS транзитом через точку доступа;

- между разными сетями BSS через две точки доступа и распределенную систему;

- между сетью BSS и проводной локальной сетью через точку доступа, распределенную систему и портал.

В сетях *с расширенным набором услуг* некоторые станции сети являются базовыми, или, в терминологии 802.11, *точками доступа* (Access Point, AP). Станция, которая выполняет функции AP, является членом какой-нибудь сети BSS (рисунок 2.40). Все базовые станции сети связаны между собой с помощью распределенной системы (Distribution System, DS), в качестве которой может использоваться та же среда (то есть радио- или инфракрасные волны), что и среда взаимодействия между станциями, или же отличная от нее, например проводная. Точки доступа вместе с распределенной системой поддерживают *службу распределенной системы* (Distribution System Service, DSS). Задачей DSS является передача пакетов между станциями, которые по каким-то причинам не могут или не хотят взаимодействовать между собой непосредственно. Наиболее очевидной причиной использования — DSS является принадлежность станций разным сетям BSS. В этом случае они передают кадр своей точке доступа, которая через DS передает его точке доступа, обслуживающей сеть BSS со станцией назначения.

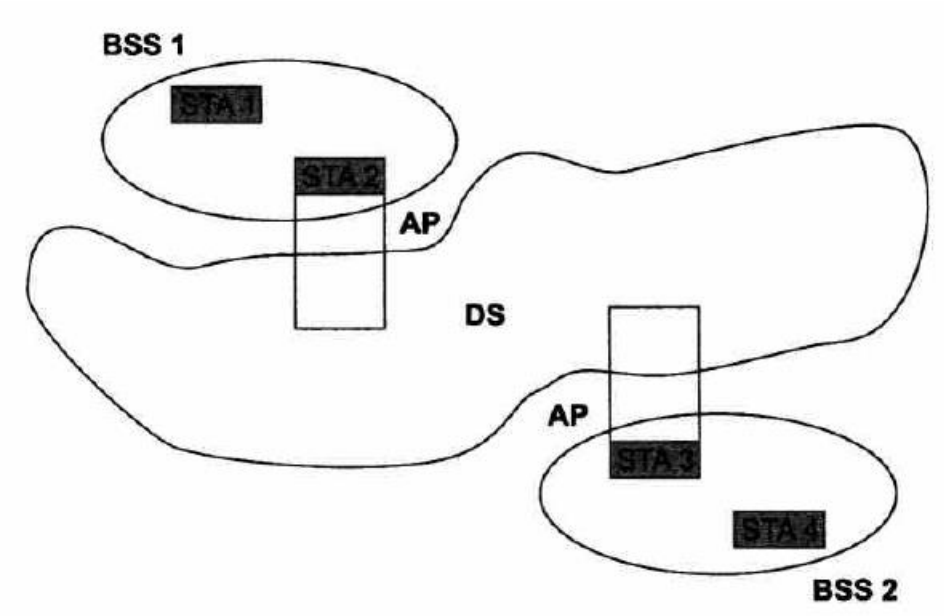


Рисунок 2.40 – Сеть с расширенным набором услуг

Сеть с *расширенным набором услуг* (Extended Service Set, ESS) состоит из нескольких сетей BSS, объединенных распределенной средой.

Сеть ESS обеспечивает станциям мобильность — они могут переходить из одной сети BSS в другую. Эти перемещения обеспечиваются функциями уровня MAC рабочих и базовых станций, поэтому они совершенно прозрачны для уровня LLC. Сеть ESS может также взаимодействовать с проводной локальной сетью.

Стек протоколов IEEE 802.11. Естественно, что стек протоколов стандарта IEEE 802.11 соответствует общей структуре стандартов комитета 802, то есть состоит из физического уровня и уровня MAC, поверх которых работает уровень LLC. Как и у всех технологий семейства 802, технология 802.11 определяется нижними двумя уровнями, то есть физическим уровнем и уровнем MAC, а уровень LLC выполняет свои стандартные функции, общие для всех технологий LAN.

Структура стека протоколов IEEE 802.11 показана на рисунке 2.41.

Уровень MAC выполняет в беспроводных сетях больше функций, чем в проводных. Функции уровня MAC в стандарте 802.11 включают:

- доступ к разделяемой среде;
- обеспечение мобильности станций при наличии нескольких базовых станций;
- обеспечение безопасности, эквивалентной безопасности проводных локальных сетей.

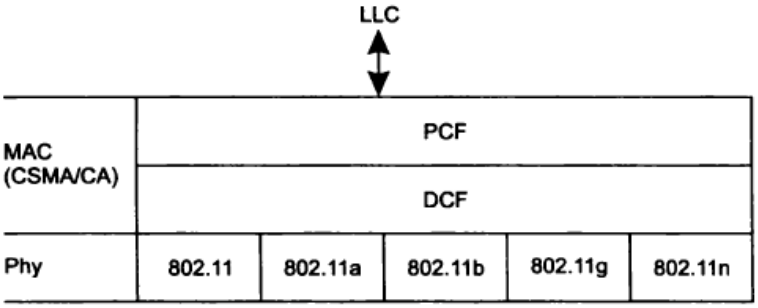


Рисунок 2.41 – Стек протоколов IEEE 802.11

В сетях 802.11 уровень MAC поддерживает два режима доступа к разделяемой среде: распределенный режим (Distributed Coordination Function, DCF) и централизованный режим (Point Coordination Function, PCF). Режим PCF применяется в тех случаях, когда необходимо приоритезировать чувствительный к задержкам трафик.

На физическом уровне существует несколько вариантов спецификаций, которые отличаются используемым частотным диапазоном, методом кодирования и, как следствие, скоростью передачи данных. Все варианты физического уровня работают с одним и тем же алгоритмом уровня MAC, но некоторые временные параметры уровня MAC зависят от используемого физического уровня.

Распределенный режим доступа. Рассмотрим сначала, как обеспечивается доступ в распределенном режиме DCF. В этом режиме реализуется метод CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance - метод прослушивания несущей частоты с множественным доступом и предотвращением коллизий). Вместо неэффективного в беспроводных сетях прямого распознавания коллизий по методу CSMA/CD здесь они выявляются косвенно. Для этого каждый переданный кадр должен подтверждаться кадром положительной квитанции,

посылаемым станцией назначения. Если же по истечении оговоренного тайм-аута квитанция не поступает, станция-отправитель считает, что произошла коллизия.

Режим DCF требует синхронизации станций. Она достигается за счет того, что временные интервалы начинают отсчитываться от момента окончания передачи очередного кадра (рис. 11.15). Это не требует передачи каких-либо специальных синхронизирующих сигналов.

Станция, которая хочет передать кадр, обязана предварительно прослушать среду. Как только она фиксирует окончание передачи кадра, она обязана отсчитать интервал времени, равный межкадровому интервалу (Inter-Frame Space, IFS). Если после истечения IFS среда все еще свободна, то начинается отсчет слотов фиксированной длительности. Кадр можно начать передавать только в начале какого-либо из слотов при условии, что среда свободна. Станция выбирает для передачи слот на основании усеченного экспоненциального двоичного алгоритма отсрочки, аналогичного используемому в методе CSMA/CD. Номер слота выбирается как случайное целое число, равномерно распределенное в интервале [0, CW], где CW означает Contention Window (конкурентное окно).

О том, как выбираются размер слота и величина конкурентного окна, рассказывается немного позже, а сейчас рассмотрим этот довольно непростой метод доступа на примере (рисунок 2.42).

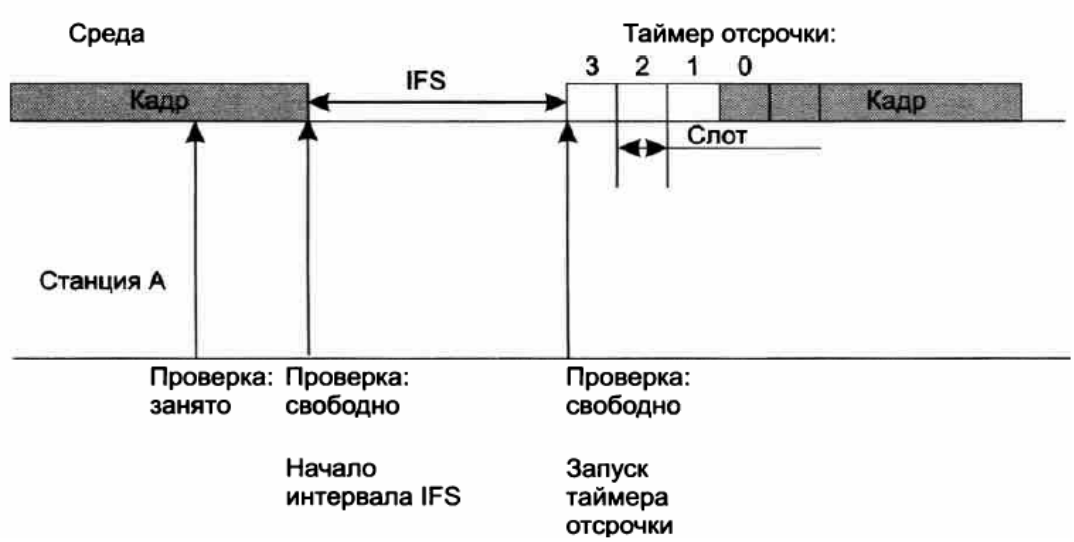


Рисунок 2.42 – Распределенный режим доступа (DCF)

Пусть станция А на основании усеченного экспоненциального двоичного алгоритма отсрочки выбрала для передачи слот 3. При этом она присваивает таймеру отсрочки (назначение которого будет ясно из дальнейшего описания) значение 3 и начинает проверять состояние среды в начале каждого слота. Если среда свободна, то из значения таймера отсрочки вычитается 1, и если результат равен нулю, то начинается передача кадра.

Таким образом обеспечивается условие незанятости всех слотов, включая выбранный. Это условие является необходимым для начала передачи.

Если же в начале какого-нибудь слота среда оказывается занятой, то вычитания единицы не происходит и таймер «замораживается». В этом случае станция начинает новый цикл доступа к среде. Как и в предыдущем цикле, станция следит за средой и при ее освобождении делает паузу в течение межкадрового интервала. Если среда осталась свободной, то станция использует значение «замороженного» таймера в качестве номера слота и выполняет описанную процедуру проверки свободных слотов с вычитанием единиц начиная с замороженного значения таймера отсрочки.

Размер слота выбирается таким образом, чтобы он превосходил время распространения сигнала между любыми двумя станциями сети плюс время, затрачиваемое станцией на распознавание ситуации занятости среды. Размер слота зависит от способа кодирования сигнала.

Если такое условие соблюдается, то каждая станция сети сумеет правильно распознать начало передачи кадра при прослушивании слотов, предшествующих слоту, выбранному ею для передачи. Это, в свою очередь, означает следующее.

Коллизия может случиться только в том случае, когда несколько станций выбирают один и тот же слот для передачи.

В этом случае кадры искажаются и квитанции подтверждения приема от станций назначения не приходят. Не получив в течение определенного времени квитанцию, отправители фиксируют факт коллизии и пытаются передать свои кадры снова. При каждой повторной неудачной попытке передачи кадра интервал [O, CW], из которого выбирается номер слота, удваивается. Если, например, начальный размер окна выбран равным 8 (то есть CW = 7), то после первой коллизии размер окна должен быть равен 16 (CW = 15), после второй последовательной коллизии - 32 и т. д. Начальное значение CW в соответствии со стандартом 802.11 должно выбираться в зависимости от типа физического уровня, используемого в беспроводной локальной сети.

Как и в методе CSMA/CD, в данном методе количество неудачных попыток передачи одного кадра ограничено, но стандарт 802.11 не дает точного значения этого верхнего предела. Когда верхний предел в N попыток достигнут, то кадр отбрасывается, а счетчик последовательных коллизий устанавливается в нуль. Этот счетчик также устанавливается в нуль, если кадр после некоторого количества неудачных попыток все же передается успешно.

В режиме DFC применяются меры для устранения эффекта скрытого терминала. Для этого станция, которая хочет захватить среду и в соответствии с описанным алгоритмом начинает передачу кадра в определенном слоте, вместо кадра данных сначала посылает станции назначения короткий служебный кадр RTS (Request To Send - запрос на передачу). На этот запрос станция назначения должна ответить служебным кадром CTS (Clear To Send - свободна для передачи), после чего станция-отправитель посылает кадр данных. Кадр CTS должен оповестить о захвате среды те станции, которые находятся вне зоны сигнала станции-отправителя, но в зоне досягаемости станции-получателя, то есть являются скрытыми терминалами для станции-отправителя.

Централизованный режим доступа. В том случае, когда в сети BSS имеется станция, выполняющая функции точки доступа, может применяться также централизованный режим доступа (PCF), обеспечивающий приоритетное обслуживание трафика. В этом случае говорят, что точка доступа играет роль арбитра среды.

Режим PCF в сетях 802.11 сосуществует с режимом DCF. Оба режима координируются с помощью трех типов межкадровых интервалов (рисунок 2.43).

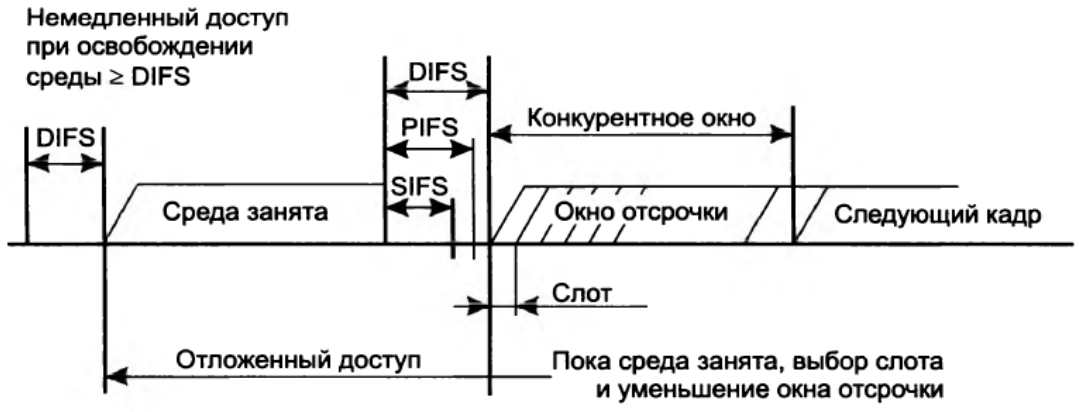


Рисунок 2.43 – Сосуществование режимов PCF и DCF

После освобождения среды каждая станция отсчитывает время простоя среды, сравнивая его с тремя значениями:

- короткий межкадровый интервал (Short IFS, SIFS);
- межкадровый интервал режима PCF (PIFS);
- межкадровый интервал режима DCF (DIFS).

Захват среды с помощью распределенной процедуры режима DCF возможен только в том случае, когда среда свободна в течение времени, равного или большего, чем DIFS. То есть в качестве IFS в режиме DCF нужно использовать интервал DIFS — самый длительный период из трех возможных, что дает этому режиму самый низкий приоритет. Межкадровый интервал SIFS имеет наименьшее значение, он служит для первоочередного захвата среды ответными кадрами CTS или квитанциями, которые продолжают или завершают уже начавшуюся передачу кадра.

Значение межкадрового интервала PIFS больше, чем SIFS, но меньше, чем DIFS. Промежутком времени между завершением PIFS и DIFS пользуется арбитр среды. В этом промежутке он может передать специальный кадр, который говорит всем станциям, что начинается контролируемый период. Получив этот кадр, станции, которые хотели бы воспользоваться алгоритмом DCF для захвата среды, уже не могут этого сделать, они должны дожидаться окончания контролируемого периода. Длительность этого периода объявляется в специальном кадре, но этот период может закончиться и раньше, если у станций нет чувствительного к задержкам трафика. В этом случае арбитр передает служебный кадр, после которого по истечении интервала DIFS начинает работать режим DCF.

На управляемом интервале реализуется централизованный метод доступа (PCF). Арбитр выполняет процедуру опроса, чтобы по очереди предоставить каждой такой станции право на использование среды, направляя ей специальный кадр. Станция, получив такой кадр, может ответить другим кадром, который подтверждает прием специального кадра и одновременно передает данные (либо по адресу арбитра для транзитной передачи, либо непосредственно станции).

Для того чтобы какая-то доля среды всегда доставалась асинхронному трафику, длительность контролируемого периода ограничена. После его окончания арбитр передает соответствующий кадр и начинается неконтролируемый период.

Каждая станция может работать в режиме PCF, для этого она должна подписаться на эту услугу при присоединении к сети.

11.6.3 Физические уровни стандарта 802.11

С момента принятия первой версии стандарта 802.11 в 1997 году одной из главных проблем, над которой работали специалисты, занимающиеся развитием беспроводных локальных сетей, была проблема повышения скорости передачи данных, чтобы приложения, хорошо работающие в проводных сетях, при переходе на беспроводную связь значительно не деградировали.

Другой немаловажной проблемой был выбранный диапазон частот радиоспектра. В соответствии с рекомендациями ITU диапазоны 2,4, 3,6 и 5 ГГц отведены для беспроводной передачи данных, при этом лицензирование этих диапазонов не рекомендуется. В разных странах существуют различные правила выбора этих диапазонов (причем правила для каждого из диапазонов могут быть разными), от свободного использования до обычного лицензирования. Помимо беспроводных локальных сетей в этих диапазонах могут работать и другие типы устройств, например, любительское радио или беспроводные сети городов.

Физические уровни стандарта 802.11 1997 года. В 1997 году комитетом 802.11 был принят стандарт, который определял функции уровня MAC вместе с тремя вариантами физического уровня, которые обеспечивают передачу данных со скоростями 1 и 2 Мбит/с.

В первом варианте средой являются инфракрасные волны диапазона 850 нм, которые генерируются либо полупроводниковым лазерным диодом, либо светодиодом (LED). Так как инфракрасные волны не проникают через стены, область покрытия LAN ограничивается зоной прямой видимости. Стандарт предусматривает три варианта распространения излучения: ненаправленную антенну, отражение от потолка и фокусное направленное излучение. В первом случае узкий луч рассеивается с помощью системы линз. Фокусное направленное излучение предназначено для организации двухточечной связи, например между двумя зданиями.

Во втором варианте в качестве передающей среды используется микроволновый диапазон 2,4 ГГц. Этот вариант основан на методе FHSS. В методе FHSS каждый узкий канал имеет ширину 1 МГц. Частотная манипуляция

(FSK) с двумя состояниями сигнала (частотами) дает скорость 1 Мбит/с, с четырьмя состояниями — 2 Мбит/с. В случае FHSS сеть может состоять из сот, причем для исключения взаимного влияния в соседних сотах могут применяться ортогональные последовательности частот. Количество каналов и частота переключения между каналами настраиваются, так что при развертывании беспроводной локальной сети можно учитывать особенности регулирования спектра частот конкретной страны.

Третий вариант, в котором используется тот же микроволновый диапазон, основан на методе DSSS, где в качестве последовательности чипов применяется 11-битный код 10110111000. Каждый бит кодируется путем двоичной фазовой (1 Мбит/с) или квадратурной фазовой (2 Мбит/с) модуляции.

Физические уровни стандартов 802.11a и 802.11b. В 1999 году были приняты два варианта стандарта физического уровня: 802.11a и 802.11b, заменяющие спецификации физического уровня 802.11 редакции 1997 года.

В спецификации 802.11b института IEEE по-прежнему используется диапазон 2,4 ГГц. Для повышения скорости до 11 Мбит/с, которая сопоставима со скоростью классического стандарта Ethernet, здесь применяется более эффективный вариант метода DSSS, опирающийся на технику Complementary Code Keying (CCK), заменившую коды Баркера.

Однако диапазон 2,4 ГГц с шириной полосы примерно в 80 МГц используется стандартом 802.11b, отличным от стандарта 1997 года способом. Этот диапазон разбит на 14 каналов, каждый из которых, кроме последнего, отстоит от соседей на 5 МГц (рисунок 2.44).

Для передачи данных согласно стандарту 802.11b используется полоса частот шириной в 22 МГц, поэтому одного канала шириной в 5 МГц оказывается недостаточно, приходится объединять несколько соседних каналов. Для того чтобы гарантировать некоторый минимум взаимных помех, возникающих от передатчиков, работающих в диапазоне 2,4 ГГц, комитет 802.11 определил так называемую спектральную маску, определяющую разрешенный спектр мощности передатчика, работающего в каком-либо из каналов. Этот спектр должен затухать не меньше чем на 30 дБ на расстоянии 11 МГц от центра канала, что и создает укрупненную полосу шириной в 22 МГц с центром в некотором из 14 каналов.

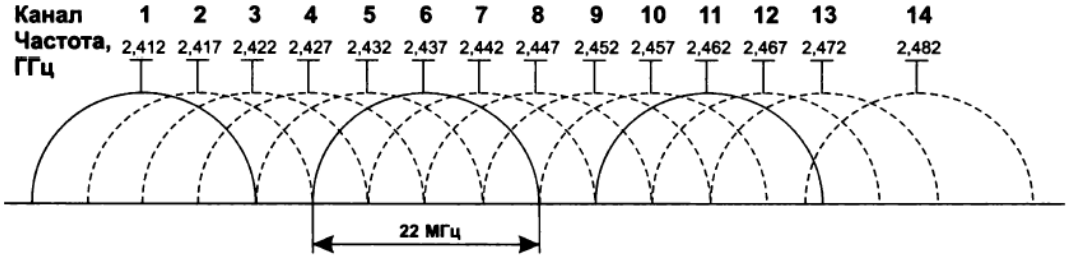


Рисунок 2.44 – Разбиение диапазона 2,4 ГГц на каналы

В результате одновременно в одной и той же области покрытия могут работать несколько независимых беспроводных сетей стандарта 802.11b. Такое использование каналов типично для США, где частотные каналы 12, 13 и 14 для сетей стандарта 802.11 не разрешены. В Европе в конце 90-х годов действовали более жесткие ограничения, например, в Испании были разрешены только каналы 10 и 11, а во Франции — только каналы 10, 11, 12 и 13, но постепенно эти ограничения были сняты, и сейчас лишь канал 14 в большинстве стран по-прежнему не задействован. Таким образом, в странах Европы максимальное количество независимых сетей, работающих в одной области покрытия, достигает четырех; обычно они занимают каналы 1, 5,9 и 13.

Оборудование стандарта 802.11b может конфигурироваться для любого из 14 каналов диапазона 2,4 ГГц, так что при возникновении помех на определенном канале можно перейти на другой.

Спецификация 802.11a обеспечивает повышение скорости передачи данных за счет использования полосы частот шириной 300 МГц из диапазона частот 5 ГГц. Так как полоса частот, отведенная для беспроводных локальных сетей, в этом диапазоне шире, то и количество каналов шириной в 5 МГц здесь больше, чем в диапазоне 2,4 ГГц, — в зависимости от правил регулирования конкретной страны их может быть 48 и более. Для передачи данных в технологии задействована полоса частот шириной 20 Мгц, что дает возможность иметь 12 и более независимых сетей в одной области покрытия.

Для передачи данных в стандарте 802.11a используется техника ортогонального частотного мультиплексирования (OFDM). Данные первоначально кодируются на 52 первичных несущих частотах методом BPSK, QPSK, 16-QAM или 64-QAM, а затем сворачиваются в общий сигнал с шириной спектра в 20 МГц. Скорость передачи данных в зависимости от метода кодирования первичной несущей частоты составляет 6,9,12,18,24,36,48 или 54 Мбит/с.

Диапазон 5 ГГц в спецификации 802.11a пока меньше ≪населен≫ и предоставляет больше частотных каналов для передачи данных. Однако его использование связано с несколькими проблемами. Во-первых, оборудование для этих частот пока еще слишком дорогое, во-вторых, в некоторых странах частоты этого диапазона подлежат лицензированию, в-третьих, волны этого диапазона хуже проходят через препятствия.

Физический уровень стандарта 802.11g. Стандарт 802.11g для физического уровня разработан рабочей группой института IEEE летом 2003 года. Он быстро завоевал популярность, так как обеспечивал те же скорости, что и стандарт 802.11a, то есть до 54 Мбит/с, но в диапазоне 2,4 ГГц, то есть в том диапазоне, где до этого удавалось достигать максимальной скорости в 11 Мбит/с на оборудовании стандарта 802.11b. В то же время стоимость оборудования стандарта 802.11g достаточно быстро стала соизмеримой со стоимостью оборудования стандарта 802.11b, что и стало причиной роста популярности новой

спецификации. В ней, так же как и в спецификации 802.11a, используется ортогональное частотное мультиплексирование (OFDM).

Диаметр сети зависит от многих параметров, в том числе от используемого диапазона частот. Обычно диаметр беспроводной локальной сети находится в пределах от 100 до 300 м вне помещений и от 30 до 40 м внутри помещений.

Физический уровень стандарта 802.11n. Стандарт 802.1n был принят в октябре 2009 года. Основной его особенностью является дальнейшее повышение скорости передачи данных (до 600 Мбит/с). Оборудование стандарта 802.11n может работать как в диапазоне 5 ГГц, так и в диапазоне 2,4 ГГц, хотя рекомендуемым диапазоном является диапазон 5 ГГц благодаря большему числу доступных каналов и меньшей интерференции с многочисленным оборудованием, работающим сегодня в диапазоне 2,4 ГГц.

Для достижения высоких скоростей в технологии 802.1 In применено несколько новых механизмов. Улучшенное кодирование OFDM и сдвоенные частотные каналы. Вместо каналов с полосой в 20 МГц, которые использовались в технологиях 802.11a и 802.11g, в технологии 802.11n применены каналы с полосой 40 МГц (для обратной совместимости допускается также работать с каналами 20 МГц). Само по себе расширение полосы в два раза должно приводить к повышению битовой скорости в два раза, но выигрыш здесь больше за счет усовершенствований в кодировании OFDM: вместо 52 первичных несущих частот на полосу в 20 МГц здесь используется 57 таких частот, а на полосу в 40 Мгц соответственно 114. Это приводит к повышению битовой скорости с 54 до 65 Мбит/с для каналов 20 Мгц и до 135 Мбит/с для каналов 40 МГц.

Уменьшение межсимвольного интервала. Для надежного распознавания кодовых символов в технологиях 80.1 la/g используется межсимвольный интервал в 800 не. Технология 802.1 In позволяет передавать данные с таким же межеимвольным интервалом, а также с межеимвольным интервалом в 400 не, что повышает битовую скорость для каналов 40 МГц до 150 Мбит/с.

Применение техники MIMO (Multiple Input Multiple Output — множественные входы и выходы). Эта техника основана на использовании одним сетевым адаптером нескольких антенн с целью лучшего распознавания сигнала, пришедшего к приемнику разными путями. Обычно из-за таких эффектов распространения радиоволн, как отражение, дифракция и рассеивание, приемник получает несколько сигналов, дошедших от передатчика по разным физическим путям и имеющим, следовательно, сдвиг по фазе. До введения техники MIMO такие явления считались негативными и с ними боролись путем применения нескольких (обычно двух) антенн, из которых в каждый момент времени использовалась только одна - та, которая принимала сигнал лучшего качества. Техника MIMO принципиально изменила отношение к сигналам, пришедшим разными путями, - эти сигналы комбинируются и путем цифровой обработки из них восстанавливается исходный сигнал.

Техника MIMO не только способствует улучшению соотношения сигнал/помеха. Благодаря возможности обрабатывать сигналы, пришедшие разными путями, для каждого потока с целью создания избыточного сигнала можно передавать с помощью нескольких антенн несколько независимых потоков данных (обычно их число меньше, чем число антенн). Эта способность систем MIMO называется пространственным мультиплексированием (spatial multiplexing). Для систем MIMO принято использовать обозначение:

TxR:S.

Здесь *T* — количество передающих антенн узла, *R* — количество принимающих антенн узла, а *S* — количество потоков данных, которые пространственно мультиплексируются. Типичный системой MIMO стандарта 802.11n является система 3 x 3 : 2, то есть система с тремя передающими и тремя принимающими антеннами, которая позволяет передавать два независимых потока данных. Система MIMO 3x3:2 обеспечивает повышение битовой скорости в два раза, то есть до 300 Мбит/с для каналов 40 МГц. Стандарт 802.11n предусматривает различные варианты системы MIMO вплоть до 4 x 4 : 4, что позволяет повысить битовую скорость до 600 Мбит/с.

Физический уровень стандарта 802.11ac. Спецификация 802.11ac является развитием спецификации 802.1 In, она обеспечивает скорости передачи данных до 1 Гбит/с за счет:

- расширения полосы индивидуального канала до 80 МГц (обязательная опция) или 160 МГц (возможная опция);
- поддержки до 8 каналов MIMO;
- применения модуляции сигнала с большим числом состояний: 256QAM вместо 64QAM у стандарта 802.1 In.

Физический уровень стандарта 802.11ad. Эта версия стандарта 802.11 была создана беспроводным гигабитным альянсом (Wireless Gigabit Alliance, WiGig) в 2009-2011 годах. Стандарт отличается тем, что в нем используется частотный диапазон 60 ГГц (а также диапазоны 2,4 и 5 ГГц). В диапазоне 60 ГГц может существовать четыре канала шириной 2,16 ГГц каждый. Широкая полоса канала позволяет передавать данные с гигабитными скоростями — до 4,6 Гбит/с при наличии одного канала и до 7 Гбит/с при мультиплексированой передаче OFDM одновременно по четырем каналам.

Однако передача данных с несущей частотой 60 ГГц сталкивается с проблемой распространения сигнала — он не проходит через стены, как сигнал частот 2,4 или 5 ГГц (хотя и отражается от препятствий, что используют приемопередатчики 802.1 lad). Поэтому область покрытия сети 802.1 lad ограничена одной комнатой, в которой находятся устройства, требующие обмена данными с гигабитными скоростями, например, приемник и телевизор стандарта HD.

Приемопередатчики стандарта 802.11ad могут также работать в диапазонах 2,4 и 5 ГГц, чтобы обеспечить связь, когда невозможно использовать сигнал частоты 60 ГГц (из-за размещения узлов сети в разных помещениях или из-за слишком большого расстояния).

12 Первичные сети

12.1 Назначение и типы первичных сетей

Первичные сети предназначены для создания коммутируемой инфраструктуры, с помощью которой можно достаточно быстро и гибко организовать постоянный канал с двухточечной топологией между двумя пользовательскими устройствами, подключенными к такой сети. Первичные сети также называют транспортными сетями, так как они предоставляют только транспортные услуги, передавая пользовательскую информацию без ее изменения.

На основе каналов, образованных первичными сетями, работают наложенные компьютерные или телефонные сети. В первичных сетях применяется техника коммутации каналов. Каналы, предоставляемые первичными сетями своим пользователям, отличаются высокой пропускной способностью — обычно от 2 Мбит/с до 100 Гбит/с, большой протяженностью, составляющей сотни и тысячи километров, а также высоким качеством, что выражается в очень низком проценте битовых ошибок. Для обеспечения таких скоростей и таких расстояний в наибольшей степени подходит оптоволоконный кабель, поэтому название «оптические сети» прочно закрепилось в качестве еще одного названия первичных сетей, несмотря на то что оптоволоконные кабели применяются и в других типах сетей, например, в сетях Ethernet.

Существует несколько поколений технологий первичных сетей:

- *плезиохронная цифровая иерархия (Plesiochronous Digital Hierarchy, PDH);*
- *синхронная цифровая иерархия (Synchronous Digital Hierarchy, SDH)* — этой технологии в Америке соответствует стандарт SONET;
- *уплотненное волновое мультиплексирование (Dense Wave Division Multiplexing, DWDM);*
- *оптические транспортные сети (Optical Transport Network, OTN)* — данная технология определят способы передачи данных по волновым каналам DWDM.

В первых двух технологиях (PDH и SDH) для разделения высокоскоростного канала применяется *временное мультиплексирование (TDM)*, а данные передаются в цифровой форме. Каждая из них поддерживает иерархию скоростей, так что пользователь может выбрать подходящую ему скорость для каналов, с помощью которых он будет строить наложенную сеть.

Технология SDH обеспечивает более высокие скорости, чем PDH, так что при построении крупной первичной сети ее магистраль строится на технологии SDH, а сеть доступа — на технологии PDH.

Сети DWDM не являются собственно цифровыми сетями, так как предоставляют своим пользователям выделенную световую волну для передачи информации, которую те могут применять по своему усмотрению и передавать по ней данные как в аналоговой, так и в дискретной (цифровой) форме. Техника мультиплексирования DWDM существенно повысила пропускную способность современных телекоммуникационных сетей, так как она позволяет организовать в одном оптическом волокне несколько десятков волновых каналов, каждый из которых может переносить информацию независимо от других каналов.

В начальный период развития технологии DWDM волновые каналы использовались в основном для передачи сигналов SDH, то есть мультиплексоры DWDM были одновременно и мультиплексорами SDH для каждого из своих волновых каналов.

Впоследствии для более эффективного использования волновых каналов DWDM была разработана технология OTN, которая позволяет передавать по волновым каналам сигналы любых технологий, включая SDH, Gigabit Ethernet, 10G Ethernet и 100G Ethernet.

12.2 Сети PDH

Технология плезиохронной цифровой иерархии (PDH) была разработана в конце 60-х годов компанией AT&T для решения проблемы связи крупных коммутаторов телефонных сетей между собой. Линии связи FDM, применяемые ранее для решения этой задачи и поддерживающие передачу голоса в аналоговой форме, исчерпали свои возможности в плане организации высокоскоростной многоканальной связи по одному кабелю. В технологии FDM по витой медной паре одновременно передавалось только 12 абонентских каналов, так что для передачи между телефонными станциями большего количества абонентских каналов приходилось прокладывать кабели с увеличенным количеством пар проводов или более дорогие коаксиальные кабели.

Иерархия скоростей. Начало технологии PDH было положено разработкой мультиплексора T-1, который позволял в цифровом виде мультиплексировать, передавать и коммутировать (на постоянной основе) голосовой трафик 24 абонентов. Так как абоненты по-прежнему пользовались обычными телефонными аппаратами, то есть передача голоса шла в аналоговой форме, то мультиплексоры T-1 сами осуществляли оцифровывание голоса с частотой 8000 Гц и кодировали голос методом импульсно-кодовой модуляции. В результате каждый абонентский канал образовывал цифровой поток данных 64 Кбит/с, а мультиплексор T-1 обеспечивал передачу на скорости 1,544 Мбит/с.

В качестве средств мультиплексирования при соединении крупных телефонных станций каналы T-1 имели недостаточную пропускную способность, поэтому была реализована идея образования каналов с иерархией скоростей. Четыре канала типа T-1 объединили в канал следующего уровня цифровой иерархии — T-2, передающий данные со скоростью 6,312 Мбит/с. Канал T-3,

образованный путем объединения семи каналов Т-2, имеет скорость 44,736 Мбит/с. Канал Т-4 объединяет 6 каналов Т-3, в результате его скорость равна 274 Мбит/с. Описанная технология получила название системы Т-каналов.

С середины 70-х годов выделенные каналы, построенные на основе систем Т-каналов, стали сдаваться телефонными компаниями в аренду на коммерческих условиях, перестав быть внутренней технологией этих компаний. Системы Т-каналов позволяют передавать не только голос, но и любые данные, представленные в цифровой форме: компьютерные данные, телевизионное изображение, факсы и т. п.

Технология систем Т-каналов была стандартизована Американским национальным институтом стандартов (ANSI), а позже - международной организацией ITU-T. При стандартизации она получила название плезиохронной цифровой иерархии (PDH). В результате внесенных ITU-T изменений возникла несовместимость американской и международной версий стандарта PDH. Аналогом систем Т-каналов в международном стандарте являются каналы типа Е-1, Е-2 и Е-3 с отличающимися скоростями - соответственно 2,048, 8,488 и 34,368 Мбит/с. Американская версия сегодня помимо США распространена также в Канаде и Японии (с некоторыми различиями), в Европе же применяется международный стандарт ITU-T. Несмотря на различия, в американской и международной версиях технологии цифровой иерархии принято использовать одни и те же обозначения для иерархии скоростей - DSn (Digital Signal п). В таблице 2.8 приводятся значения для всех введенных стандартами уровней скоростей обеих технологий.

Таблица 2.8 – Иерархия цифровых скоростей

Америка				**ITU-T (Европа)**		
обозначение скорости	**количество голосовых каналов**	**количество каналов предыдущего уровня**	**скорость, Мбит/с**	**количество голосовых каналов**	**количество каналов предыдущего уровня**	**скорость, Мбит/с**
DS-0	1	1	64 Кбит/с	1	1	64 Кбит/с
DS-1	24	24	1,544	30	30	2,048
DS-2	96	4	6,312	120	4	8,488
DS-3	672	7	44,736	480	4	34,368
DS-4	4032	6	274,176	1920	4	139,264

На практике в основном используются каналы Т-1/Е-1 и Т-3/Е-3.

Методы мультиплексирования

Мультиплексор Т-1 обеспечивает передачу данных 24 абонентов со скоростью 1,544 Мбит/с в кадре, имеющем достаточно простой формат. В этом кадре последовательно передается по одному байту каждого абонента, а после 24 байт вставляется один бит синхронизации. Первоначально устройства Т-1 функционировали только на внутренних тактовых генераторах и каждый кадр с помощью битов синхронизации мог передаваться асинхронно.

Сегодня мультиплексоры и коммутаторы первичной сети PDH работают на централизованной тактовой частоте, распределяемой из одной или нескольких точек сети.

Однако принцип формирования кадра не изменился, поэтому биты синхронизации в кадре по-прежнему присутствуют. Суммарная скорость пользовательских каналов составляет 24 x 64 = 1,536 Мбит/с, а еще 8 Кбит/с добавляют биты синхронизации, итого получается 1,544 Мбит/с.

Теперь рассмотрим еще одну особенность формата кадра Т-1. В аппаратуре Т-1 восьмой бит каждого байта в кадре имеет назначение, зависящее от типа передаваемых данных и поколения аппаратуры. При передаче голоса с помощью этого бита переносится служебная информация, к которой относятся номер вызываемого абонента и другие сведения, необходимые для установления соединения между абонентами сети. Протокол, обеспечивающий такое соединение, называется в телефонии сигнальным протоколом. Поэтому реальная скорость передачи пользовательских данных в этом случае составляет не 64, а 56 Кбит/с. Техника применения восьмого бита для служебных целей получила название «кражи бита».

Версия технологии PDH, описанная в международных стандартах G.700-G.706 ITU-T, как уже отмечалось, имеет отличия от американской технологии систем Т-каналов. В частности, в ней не используется схема «кражи бита». При переходе к следующему уровню иерархии коэффициент кратности скорости имеет постоянное значение 4. Вместо восьмого бита в канале Е-1 на служебные цели отводятся 2 байта из 32, а именно нулевой (для целей синхронизации приемника и передатчика) и шестнадцатый (в нем передается служебная сигнальная информация). Для голосовых или компьютерных данных остается 30 каналов со скоростью передачи 64 Кбит/с каждый.

При мультиплексировании нескольких пользовательских потоков в мультиплексорах PDH применяется техника, называемая бит-стаффингом. К этой технике прибегают, когда скорость пользовательского потока оказывается несколько меньше, чем скорость объединенного потока, - подобные проблемы могут возникать в сети, состоящей из большого количества мультиплексоров, несмотря на все усилия по централизованной синхронизации узлов сети (в природе нет ничего идеального, в том числе идеально синхронных узлов сети). В результате мультиплексор PDH периодически сталкивается с ситуацией, когда ему «не хватает» бита для представления в объединенном потоке того или иного пользовательского потока. В этом случае мультиплексор просто вставляет в объединенный поток бит-вставку и отмечает этот факт в служебных битах объединенного кадра. При демультиплексировании объединенного потока бит-вставка удаляется из пользовательского потока, который возвращается в исходное состояние. Техника бит-стаффинга применяется как в международной, так и в американской версии PDH.

Отсутствие полной синхронности потоков данных при объединении низкоскоростных каналов в высокоскоростные и дало название технологии PDH («плезиохронный» означает «почти синхронный»).

Синхронизация сетей PDH. В случае небольшой сети PDH, например, сети города, синхронизация всех устройств сети из одной точки представляется достаточно простым делом. Однако для более крупных сетей, например, сетей масштаба страны, которые состоят из некоторого количества региональных сетей, синхронизация всех устройств сети представляет собой проблему. Общий подход к решению этой проблемы описан в стандарте ITU-T G.810. Он заключается в организации в сети иерархии эталонных источников синхросигналов, а также системы распределения синхросигналов по всем узлам сети (рисунок 2.45).

Каждая крупная сеть должна иметь по крайней мере один первичный эталонный генератор (ПЭГ) синхросигналов (в англоязычном варианте — Primary Reference Clock, PRC).

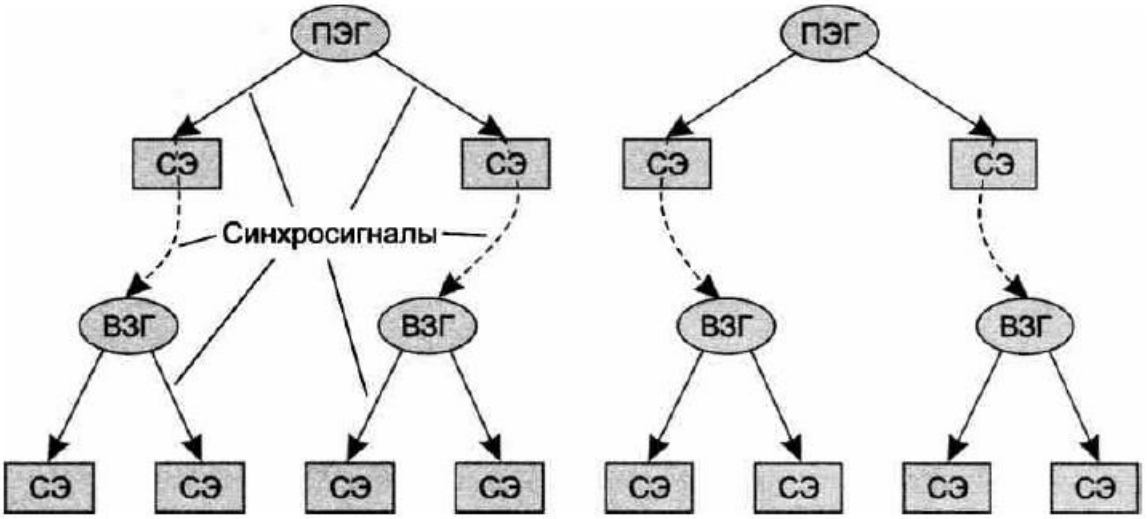


Рисунок 2.45 – Организация распределения синхросигналов по узлам сети PDH

Это очень точный источник синхросигналов, способный вырабатывать синхросигналы с относительной точностью частоты не хуже 10^{-11} (такую точность требуют стандарты ITU-T G.811 и ANSI T1.101, в последнем для описания точности ПЭГ применяется название Stratum 1). На практике в качестве ПЭГ используют либо автономные атомные (водородные или цезиевые) часы, либо часы, синхронизирующиеся от спутниковых систем точного мирового времени, таких как GPS или ГЛОНАСС. Обычно точность ПЭГ достигает 10^{-13}.

Стандартным синхросигналом является сигнал тактовой частоты уровня DS1, то есть частоты 2048 кГц для международного варианта стандартов PDH и 1544 кГц для американского варианта этих стандартов.

Синхросигналы от ПЭГ непосредственно поступают на специально отведенные для этой цели синхровходы магистральных устройств сети PDH. В том случае, если это составная сеть, то каждая крупная сеть, входящая в состав составной сети (например, региональная сеть, входящая в состав национальной сети), имеет свой ПЭГ.

Для синхронизации немагистральных узлов используется вторичный задающий генератор (ВЗГ) синхросигналов, который в варианте ITU-T называют

Secondary Reference Clock (SRC), а в варианте ANSI — генератором уровня Stratum 2. ВЗГ работает в режиме принудительной синхронизации, являясь ведомым таймером в паре ПЭГ-ВЗГ. Обычно ВЗГ получает синхросигналы от некоторого ПЗГ через промежуточные магистральные узлы сети, при этом для передачи синхросигналов используются биты служебных байтов кадра, например нулевого байта кадра E-1 в международном варианте PDH.

Точность ВЗГ меньше, чем точность ПЭГ: ITU-T в стандарте G.812 определяет ее как «не хуже 10^{-9}», а точность генераторов Stratum 2 должна быть не «хуже 1,6 x 10^{-8}».

Иерархия эталонных генераторов может быть продолжена, если это необходимо, при этом точность каждого более низкого уровня, естественно, понижается. Генераторы нижних уровней могут использовать для выработки своих синхросигналов несколько эталонных генераторов более высокого уровня, но при этом в каждый момент времени один из них должен быть основным, а остальные — резервными; такое построение системы синхронизации обеспечивает ее отказоустойчивость. Однако в этом случае нужно приоритезировать сигналы генераторов более высоких уровней. Кроме того, при построении системы синхронизации требуется гарантировать отсутствие петель синхронизации.

Методы синхронизации цифровых сетей, кратко описанные в этом разделе, применимы не только к сетям PDH, но и к другим сетям, работающим на основе синхронного TDM- мультиплексирования, например к сетям SDH, а также к сетям цифровых телефонных коммутаторов.

12.3 Сети SONET/SDH

Как американский, так и международный вариант технологии PDH обладает рядом недостатков, основным из которых является сложность и неэффективность операций мультиплексирования и демультиплексирования пользовательских данных. Применение техники бит-стаффинга для выравнивания скоростей потоков приводит к тому, что для извлечения пользовательских данных одного из каналов объединенного канала необходимо полностью (!) демультиплексировать кадры объединенного канала.

Также в технологии PDH не предусмотрены встроенные средства обеспечения отказоустойчивости и администрирования сети.

Наконец, недостатком PDH являются слишком низкие по современным понятиям скорости передачи данных — ее иерархия скоростей заканчивается уровнем 139 Мбит/с.

Недостатки и ограничения технологии PDH были учтены и преодолены разработчиками технологии синхронных оптических сетей (Synchronous Optical NET, SONET), первый вариант стандарта которой появился в 1984 году. Затем она была стандартизована комитетом T-1 института ANSI. Международная стандартизация технологии проходила под эгидой Европейского института телекоммуникационных стандартов (European Telecommunications Standards

Institute, ETSI) и сектором телекоммуникационной стандартизации союза ITU (ITU Telecommunication Standardization Sector, ITU-T) совместно с ANSI и ведущими телекоммуникационными компаниями Америки, Европы и Японии.

Основной целью разработчиков международного стандарта было создание технологии, способной передавать трафик всех существующих цифровых каналов уровня PDH (как американских T1-T3, так и европейских E1-E4) по высокоскоростной магистральной сети на базе волоконно-оптических кабелей и обеспечить иерархию скоростей, продолжающую иерархию технологии PDH до скорости в несколько гигабит в секунду.

В результате длительной работы ITU-T и ETSI удалось разработать и принять международный стандарт SDH (Synchronous Digital Hierarchy — синхронная цифровая иерархия). Кроме того, стандарт SONET был доработан так, чтобы аппаратура и сети SDH и SONET являлись совместимыми и могли мультиплексировать входные потоки практически любого стандарта PDH — и американского, и европейского.

Иерархия скоростей и методы мультиплексирования. Поддерживаемая технологией SONET/SDH иерархия скоростей представлена в таблице 2.8.

В стандарте SDH все уровни скоростей (и соответственно форматы кадров для этих уровней) имеют общее название STM-N (Synchronous Transport Module level N — синхронный транспортный модуль уровня N). В технологии SONET существуют два обозначения для уровней скоростей: название STS-N (Synchronous Transport Signal level N — синхронный транспортный сигнал уровня N) употребляется в случае передачи данных электрическим сигналом, а название OC-N (Optical Carrier level N - оптоволоконная линия связи уровня N) используют в случае передачи данных по волоконно-оптическому кабелю. Далее для упрощения изложения мы сосредоточимся на технологии SDH.

Таблица 2.8 – Иерархия скоростей SONET/SDH

SDH	SONET	Скорость
	STS-1, OC-1	51,84 Мбит/с
STM-1	STS-3, OC-3	155,520 Мбит/с
STM-3	OC-9	466,560 Мбит/с
STM-4	OC-12	622,080 Мбит/с
STM-6	OC-18	933,120 Мбит/с
STM-8	OC-24	1,244 Гбит/с
STM-12	OC-36	1,866 Гбит/с
STM-16	OC-48	2,488 Гбит/с
STM-64	OC-192	9,953 Гбит/с
STM-256	OC-768	39,81 Гбит/с

Кадры STM-N имеют достаточно сложную структуру, позволяющую агрегировать в общий магистральный поток потоки SDH и PDH различных скоростей, а также выполнять операции ввода-вывода без полного демультиплексирования магистрального потока.

Операции мультиплексирования и ввода-вывода выполняются при помощи виртуальных контейнеров (Virtual Container, VC), в которых блоки данных PDH можно транспортировать через сеть SDH. Помимо блоков данных PDH в виртуальный контейнер помещается еще некоторая служебная информация, в частности заголовок пути (Path OverHead, POH) контейнера, в котором размещается статистическая информация о процессе прохождения контейнера вдоль пути от его начальной до конечной точки (сообщения об ошибках), а также другие служебные данные, например, индикатор установления соединения между конечными точками. В результате размер виртуального контейнера оказывается больше, чем соответствующая нагрузка в виде блоков данных PDH, которую он переносит. Например, виртуальный контейнер VC-12 помимо 32 байт данных потока E-1 содержит еще 3 байта служебной информации.

В технологии SDH определено несколько типов виртуальных контейнеров, предназначенных для транспортировки основных типов блоков данных PDH: VC-11 (1,5 Мбит/с), VC-12 (2 Мбит/с), VC-2 (6 Мбит/с), VC3 (34/45 Мбит/с) и VC-4 (140 Мбит/с).

Виртуальные контейнеры являются единицей коммутации мультиплексоров SDH. В каждом мультиплексоре существует таблица соединений (называемая также таблицей кросс-соединений), в которой указано, например, что контейнер VC-12 порта P1 соединен с контейнером VC12 порта P5, а контейнер VC3 порта P8 – с контейнером VC3 порта P9. Таблицу соединений формирует администратор сети с помощью системы управления или управляющего терминала на каждом мультиплексоре так, чтобы обеспечить сквозной путь между конечными точками сети, к которым подключено пользовательское оборудование.

Чтобы совместить в рамках одной сети механизмы синхронной передачи кадров (STM-N) и асинхронный характер переносимых этими кадрами пользовательских данных PDH, в технологии SDH применяются указатели. Концепция указателей – ключевая в технологии SDH, она заменяет принятое в PDH выравнивание скоростей асинхронных источников посредством дополнительных битов. Указатель определяет текущее положение виртуального контейнера в агрегированной структуре более высокого уровня, каковой является трибутарный блок (Tributary Unit, TU) либо административный блок (Administrative Unit, AU). Собственно, основное отличие этих блоков от виртуального контейнера заключается в наличии дополнительного поля указателя. С помощью этого указателя виртуальный контейнер может «смещаться» в определенных пределах внутри своего трибутарного или административного блока, если скорость пользовательского потока несколько отличается от скорости кадра SDH, куда этот поток мультиплексируется.

Именно благодаря системе указателей мультиплексор находит положение пользовательских данных в синхронном потоке байтов кадров STM-N и «на лету» извлекает их оттуда, чего механизм мультиплексирования, применяемый в PDH, делать не позволяет.

Трибутарные блоки объединяются в группы, а те, в свою очередь, входят в административные блоки. Группа административных блоков (Administrative Unit

Group, AUG) в количестве N и образует полезную нагрузку кадра STM-N. Помимо этого, в кадре имеется заголовок с общей для всех блоков AU служебной информацией. На каждом шаге преобразования к предыдущим данным добавляется несколько служебных байтов: они помогают распознать структуру блока или группы блоков и затем определить с помощью указателей начало пользовательских данных.

12.4 Типы оборудования сети SDH

Основным элементом сети SDH является мультиплексор (рисунок 2.46). Обычно он оснащен некоторым количеством портов PDH и SDH: например, портами PDH на 2 и 34/45 Мбит/с и портами SDH STM-1 на 155 Мбит/с и STM-4 на 622 Мбит/с. Порты мультиплексора SDH делятся на агрегатные и трибутарные.

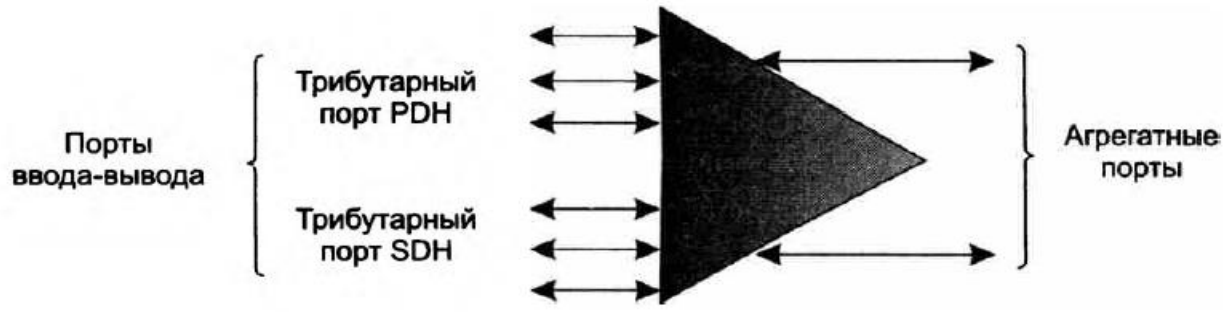


Рисунок 2.46 – Мультиплексор SDH

Трибутарные порты часто называют также портами ввода-вывода, а агрегатные — линейными портами. Эта терминология отражает типовые топологии сетей SDH, где имеется ярко выраженная магистраль в виде цепи или кольца, по которой передаются потоки данных, поступающие от пользователей сети через порты ввода-вывода (трибутарные порты), то есть втекающие в агрегированный поток («tributary» дословно означает «приток»).

Мультиплексоры SDH обычно разделяют на два типа, разница между которыми определяется положением мультиплексора в сети SDH (рисунок 2.47).

Терминальный мультиплексор (Terminal Multiplexer, TM) завершает агрегатный канал, мультиплексируя в нем большое количество трибутарных каналов, поэтому он оснащен одним агрегатным и множеством трибутарных портов.

Мультиплексор ввода-вывода (Add-Drop Multiplexer, ADM) занимает промежуточное положение на магистрали (в кольце, цепи или смешанной топологии). Он имеет два агрегатных порта, транзитом передавая агрегатный поток данных. С помощью небольшого количества трибутарных портов такой мультиплексор вводит в агрегатный поток или выводит из агрегатного потока данные трибутарных каналов.

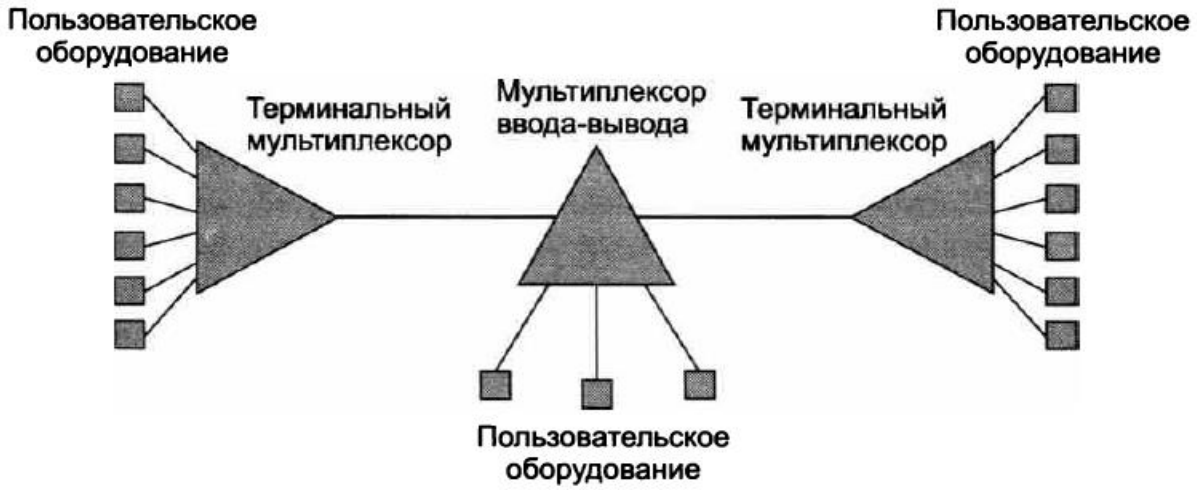


Рисунок 2.47 – Типы мультиплексоров SDH

Иногда также выделяют мультиплексоры, которые выполняют операции коммутации над произвольными виртуальными контейнерами — так называемые цифровые кроссконнекторы (Digital Cross-Connect, DXC). В таких мультиплексорах не делается различий между агрегатными и трибутарными портами, так как они предназначены для работы в ячеистой топологии, где выделить агрегатные потоки невозможно.

Помимо мультиплексоров в состав сети SDH могут входить регенераторы сигналов, необходимые для преодоления ограничений по расстоянию между мультиплексорами. Эти ограничения зависят от мощности оптических передатчиков, чувствительности приемников и затухания волоконно-оптического кабеля. Регенератор преобразует оптический сигнал в электрический и обратно, при этом восстанавливается форма сигнала и его временные характеристики. В настоящее время регенераторы SDH применяются достаточно редко, так как стоимость их ненамного ниже стоимости мультиплексора, а функциональные возможности несоизмеримо беднее.

12.5 Типовые топологии сетей SDH

В сетях SDH применяются различные топологии связей. Наиболее часто используются кольца и линейные цепи мультиплексоров, также находит все большее применение ячеистая топология, близкая к полносвязной. Кольцо SDH строится из мультиплексоров ввода-вывода, имеющих по крайней мере по два агрегатных порта (рисунок 2.48, а). Пользовательские потоки вводятся в кольцо и выводятся из кольца через трибутарные порты, образуя двухточечные соединения (на рисунке показаны в качестве примера два таких соединения). Кольцо является классической регулярной топологией, обладающей потенциальной отказоустойчивостью — при однократном обрыве кабеля или выходе из строя мультиплексора соединение сохранится, если его направить по кольцу в противоположном направлении. Кольцо обычно строится на основе кабеля с

двумя оптическими волокнами, но иногда для повышения надежности и пропускной способности применяют четыре волокна.

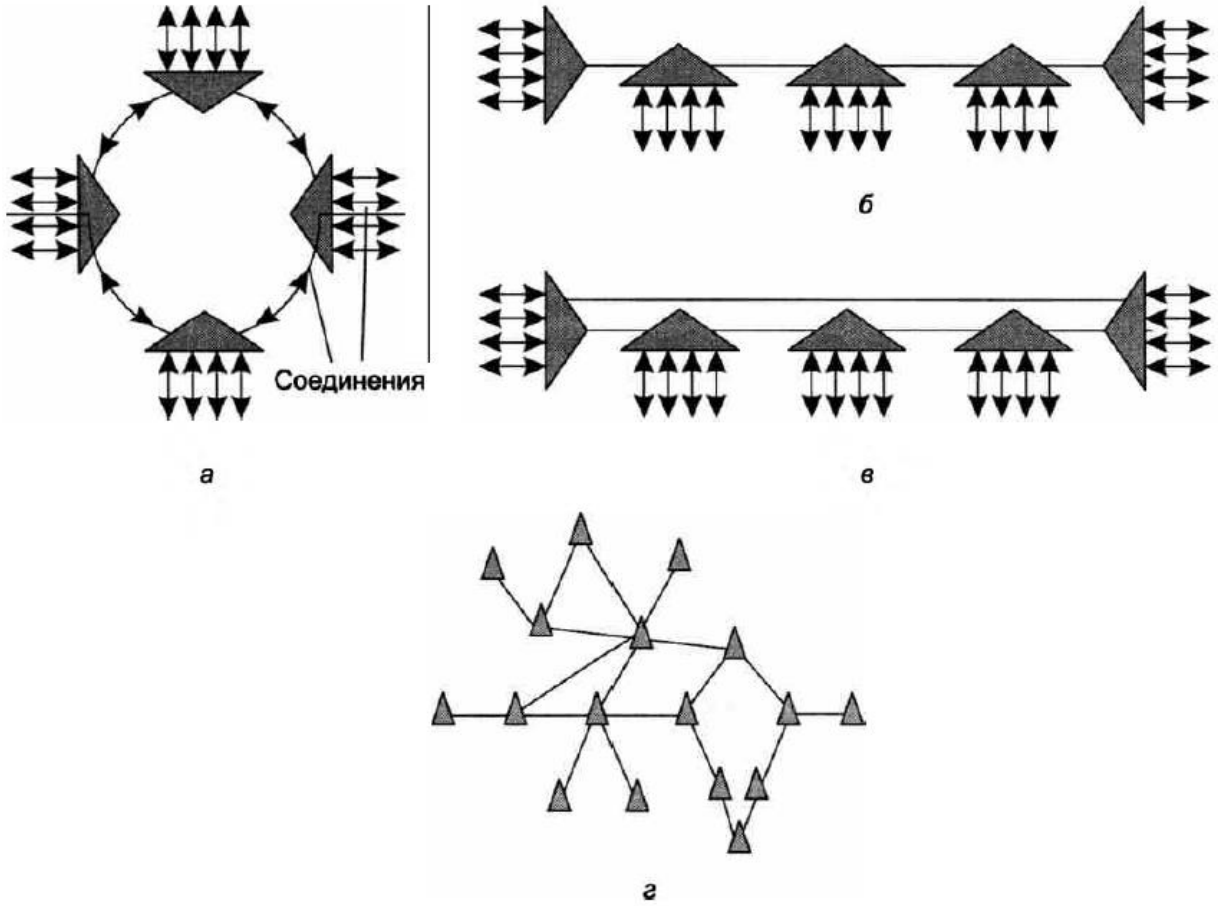


Рисунок 2.48 – Типовые топологии

Цепь (рисунок 2.48, б) — это линейная последовательность мультиплексоров, из которых два оконечных играют роль терминальных мультиплексоров, остальные — мультиплексоров ввода-вывода. Обычно сеть с топологией цепи применяется в тех случаях, когда узлы имеют соответствующее географическое расположение, например, вдоль магистрали железной дороги или трубопровода. Правда, в подобных случаях может применяться и плоское кольцо (рисунок 2.48, в), обеспечивающее более высокий уровень отказоустойчивости за счет двух дополнительных волокон в магистральном кабеле и по одному дополнительному агрегатному порту у терминальных мультиплексоров.

Эти базовые топологии могут комбинироваться при построении сложной и разветвленной сети SDH, образуя участки с радиально-кольцевой топологией, соединениями «кольцо-кольцо» и т. п. Наиболее общим случаем является ячеистая топология (рисунок 2.48, г), в которой мультиплексоры соединяются друг с другом большим количеством связей, за счет чего сеть может достичь очень высокой степени производительности и надежности.

Список использованных источников

1. С. А. Орлов. Организация ЭВМ и систем: Учебник для ВУЗов. Для бакалавров и магистров. 4-е издание. – СПб.: Питер, 2020. – 688 с.

2. Таненбаум Э. Архитектура компьютера / Э. Таненбаум, Т. Остин. – 6-е изд. – Санкт-Петербург: Питер, 2019. – 816 с.

3. Юров В. И. Assembler: Учебник / В. И. Юров. – 2-е изд. – Санкт-Петербург: Питер, 2011. – 637 с.

4. Брайант Рэндал Э., О'Халларон Дэвид Р. Компьютерные системы: архитектура и программирование. 3-е изд. / пер. с англ. А. Н. Киселева. – М.: ДМК Пресс, 2022. – 994 с.

5. Хамахер, К. Организация ЭВМ / К. Хамахер, З. Вранешич, С. Заки. – 5-е изд. – СПб. : Питер, 2003. – 848 с.

6. Калабухов Е. В. Конструирование программ и языки программирования. Язык программирования Ассемблер : пособие / Е. В. Калабухов, И. В. Лукьянова, А. Г. Третьяков. – Минск : БГУИР, 2016. − 80 с.

7. Ружицкая, Е. А. Программирование на языке Assembler : системы счисления, программная модель микропроцессора, арифметические команды : практическое пособие / Е. А. Ружицкая, О. Г. Осипова, В. А. Ковалёва ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2016. – 47 с.

8. Ружицкая, Е. А. Программирование на языке Assembler : программирование ветвящихся и циклических вычислительных процессов, обработка массивов : практическое пособие / Е. А. Ружицкая, О. Г. Осипова, В. А. Ковалёва ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2016. – 47 с.

9. Ружицкая, Е. А. Программирование на языке Assembler : BCD-числа, цепочечные команды : практическое пособие / Е. А. Ружицкая, Е. Ю. Кузьменкова ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2017. – 47 с.

10. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для ВУЗов. Юбилейное издание. – СПб.: Питер, 2020. – 1008 с.

11. Урбанович, П. П. Компьютерные сети : учебное пособие / П. П. Урбанович, Д. М. Романенко. – Москва ; Вологда : Инфра-Инженерия, 2022. – 460с.

2 ПРАКТИЧЕСКИЙ РАЗДЕЛ

2.1 ЛАБОРАТОРНЫЙ ПРАКТИКУМ. ЧАСТЬ 1. ПРОГРАММНАЯ МОДЕЛЬ АРХИТЕКТУРЫ IA-32

ОГЛАВЛЕНИЕ

1 Программная модель микропроцессора IA-32

Любая выполняющаяся программа получает в свое распоряжение определенный набор ресурсов процессора. Эти ресурсы необходимы для обработки и хранения в памяти команд и данных программы, а также информации о текущем состоянии программы и процессора. Программную модель процессора в архитектуре IA-32 процессоров Intel составляет следующий набор ресурсов (рисунок 1):

- пространство адресуемой памяти до 232 – 1 байт (4 Гбайт), для Pentium III/IV — до 236 – 1 байт (64 Гбайт);

- набор регистров для хранения данных общего назначения; набор сегментных регистров; набор регистров состояния и управления; набор регистров устройства вычислений с плавающей точкой (сопроцессора); набор регистров целочисленного MMX-расширения, отображенных на регистры сопроцессора (впервые появились в архитектуре процессора Pentium MMX); набор регистров MMX-расширения с плавающей точкой (впервые появились в архитектуре процессора Pentium III);

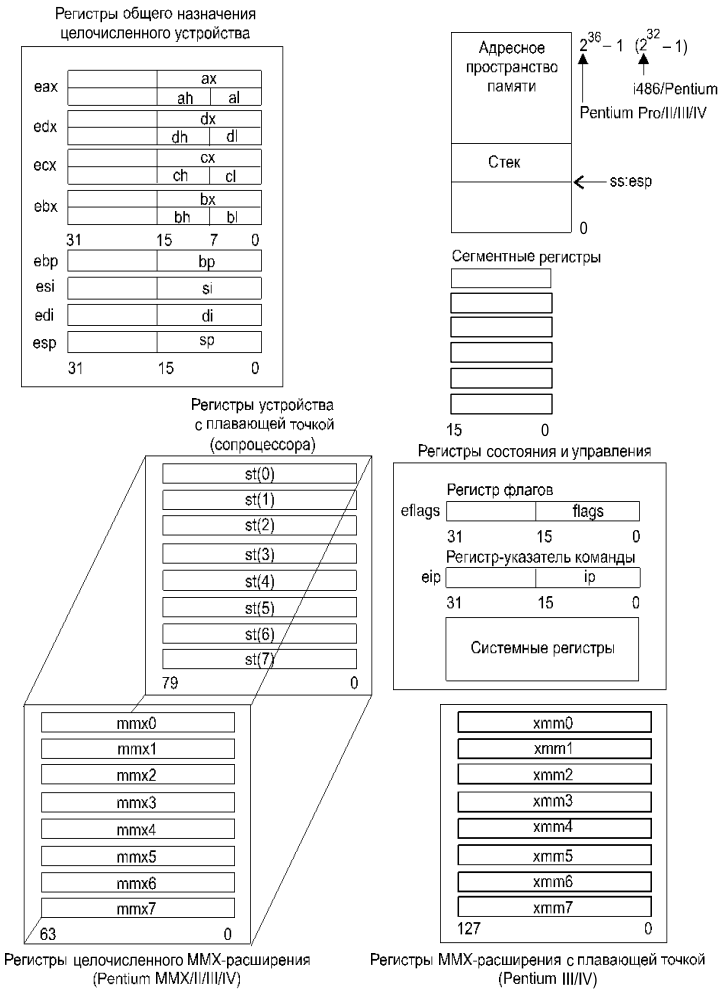


Рисунок 1 – Программная модель архитектуры IA-32

- программный стек — специальная информационная структура, работа с которой предусмотрена на уровне машинных команд (более подробно она будет обсуждена позже).

Это основной набор ресурсов. Кроме того, к ресурсам, поддерживаемым архитектурой IA-32, необходимо отнести порты ввода-вывода, счетчики мониторинга производительности.

Программные модели более ранних процессоров (i486, первые Pentium) отличаются меньшим размером адресуемого пространства оперативной памяти ($2^{32} - 1$, так как разрядность их шины адреса составляет 32 бита) и отсутствием некоторых групп регистров. Для каждой группы регистров в скобках показано, начиная с какой модели данная группа регистров появилась в программной модели процессоров Intel. Если такого обозначения нет, то это означает, что данная группа регистров присутствовала в процессорах i386 и i486. Что касается еще более ранних процессоров i8086/88, то на самом деле они тоже полностью представлены на схеме, но составляют лишь ее небольшую ее часть. В программную модель данных процессоров входят 8- и 16-разрядные регистры общего назначения, сегментные регистры, регистры FLAGS, IP и адресное пространство памяти размером до 1Мбайт. Свойства некоторых перечисленных далее программно-доступных ресурсов определяются текущим режимом работы процессора.

Набор регистров

Регистрами называются области высокоскоростной памяти, расположенные внутри процессора в непосредственной близости от его исполнительного ядра. Доступ к ним осуществляется несравнимо быстрее, чем к ячейкам оперативной памяти. Соответственно, машинные команды с операндами в регистрах выполняются максимально быстро, поэтому в программах на языке ассемблера регистры используются очень интенсивно. К сожалению, архитектура IA-32 предоставляет в распоряжение программиста не слишком много регистров, поэтому они являются критически важным ресурсом и за их содержимым приходится следить очень внимательно.

Большинство регистров имеют определенное функциональное назначение. С точки зрения программиста, их можно разделить на две большие группы.

Первую группу образуют пользовательские регистры, к которым относятся:

- регистры общего назначения EAX/AX/AH/AL, EBX/BX/BH/BL, EDX/DX/DH/DL, ECX/ CX/CH/CL, EBP/BP, ESI/SI, EDI/DI, ESP/SP предназначены для хранения данных и адресов, программист может их использовать (с определенными ограничениями) для реализации своих алгоритмов;

- сегментные регистры CS, DS, SS, ES, FS, GS используются для хранения адресов сегментов в памяти;
- регистры сопроцессора ST(0), ST(1), ST(2), ST(3), ST(4), ST(5), ST(6), ST(7) предназначены для написания программ, использующих тип данных с плавающей точкой;
- целочисленные регистры MMX-расширения MMX0, MMX1, MMX2, MMX3, MMX4, MMX5, MMX6, MMX7;
- регистры MMX-расширения с плавающей точкой XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6, XMM7;
- регистры состояния и управления (регистр флагов EFLAGS/FLAGS и регистр-указатель команды EIP/IP) содержат информацию о состоянии процессора, исполняемой программы и позволяют изменить это состояние.

Во вторую группу входят системные регистры, то есть регистры, предназначенные для поддержания различных режимов работы, сервисных функций, а также регистры, специфичные для определенной модели процессора. Перечислим системные регистры, поддерживаемые IA-32:

- управляющие регистры CR0…CR4 определяют режим работы процессора и характеристики текущей исполняемой задачи;
- регистры управления памятью GDTR, IDTR, LDTR и TR используются в защищенном режиме работы процессора для локализации управляющих структур этого режима;
- отладочные регистры DR0…DR7 предназначены для мониторинга и управления различными аспектами отладки;
- регистры типов областей памяти MTRR используются для аппаратного управления кэшированием в целях назначения соответствующих свойств областям памяти;
- машинно-зависимые регистры MSR используются для управления процессором, контроля за его производительностью, получения информации об ошибках.

Почему в обозначениях многих из регистров общего назначения присутствует наклонная разделительная черта? Это не разные регистры — это части одного большого 32-разрядного регистра, но их можно использовать в программе как отдельные объекты. Зачем так сделано? Чтобы обеспечить работоспособность программ, написанных для прежних 16-разрядных моделей процессоров фирмы Intel начиная с i8086. Процессоры i486 и Pentium имеют в основном 32-разрядные регистры.

Их количество, за исключением сегментных регистров, такое же, как и у i8086, но размерность больше, что и отражено в обозначениях — они имеют приставку e (extended).

Регистры общего назначения. Регистры общего назначения используются в программах для хранения:

- операндов логических и арифметических операций;
- компонентов адреса;

- указателей на ячейки памяти.

В принципе, все эти регистры доступны для хранения операндов без особых ограничений, хотя в определенных условиях некоторые из них имеют жесткое функциональное назначение, закрепленное на уровне логики работы машинных команд. Среди всех этих регистров особо следует выделить регистр ESP. Его не следует использовать явно для хранения каких-либо операндов программы, так как в нем хранится указатель на вершину стека программы.

Все регистры этой группы позволяют обращаться к своим «младшим» частям (см. рисунок 1.7). Обращение производится по их именам. Важно отметить, что использовать для самостоятельной адресации можно только младшие 16- и 8-разрядные части этих регистров. Старшие 16 битов как самостоятельные объекты недоступны. Это сделано, как мы отметили ранее, для совместимости с первыми 16-разрядными моделями процессоров фирмы Intel. Перечислим регистры, относящиеся к группе регистров общего назначения и физически находящиеся в процессоре внутри арифметико-логического устройства (поэтому их еще называют *регистрами АЛУ*):

регистр-аккумулятор (Accumulator register) EAX/AX/AH/AL применяется для хранения промежуточных данных, в некоторых командах его использование обязательно;

базовый регистр (Base register) EBX/BX/BH/BL применяется для хранения базового адреса некоторого объекта в памяти;

регистр-счетчик (Count register) ECX/CX/CH/CL применяется в командах, производящих некоторые повторяющиеся действия. Использование регистра-счетчика зачастую скрыто в алгоритме работы той или иной команды. Например, команда организации цикла LOOP помимо передачи управления анализирует и уменьшает на единицу значение регистра ECX/CX;

регистр данных (Data register) EDX/DX/DH/DL, так же как и регистр EAX/AX/AH/ AL, хранит промежуточные данные (в некоторых командах его явное использование обязательно, в других он используется неявно).

Следующие два регистра предназначены для поддержки так называемых цепочечных операций, то есть операций, производящих последовательную обработку цепочек элементов, каждый из которых может иметь длину 32, 16 или 8 бит:

регистр индекса источника (Source Index register) ESI/SI в цепочечных операциях содержит текущий адрес элемента в цепочке-источнике;

регистр индекса приемника (Destination Index register) EDI/DI в цепочечных операциях содержит текущий адрес в цепочке-приемнике.

В архитектуре процессора на программно-аппаратном уровне поддерживается такая структура данных, как *стек*. Для работы со стеком в системе команд процессора есть специальные команды, а в программной модели процессора для этого существуют специальные регистры:

регистр указателя стека (Stack Pointer register) ESP/SP содержит

указатель на вершину стека в текущем сегменте стека;

регистр указателя базы кадра стека (Base Pointer register) EBP/BP предназначен для организации произвольного доступа к данным внутри стека.

Не стоит пугаться столь жесткого функционального назначения регистров АЛУ. На самом деле при программировании хранить операнды команд можно в большинстве регистров, причем практически в любых сочетаниях. Однако всегда следует помнить, что некоторые команды требуют строго определенных регистров. Цель подобного жесткого закрепления регистров для этих команд — более компактная кодировка их машинного представления. Знание особенностей использования регистров машинными командами (см. приложение) позволяет, при необходимости, экономить память, занимаемую кодом программы, и более эффективно программировать алгоритм.

Сегментные регистры. Процессоры Intel аппаратно поддерживают *сегментную* организацию программы. Это означает, что любая программа состоит из трех сегментов: кода, данных и стека. Логически машинные команды в архитектуре IA-32 построены так, что при выборке каждой команды для доступа к данным программы или к стеку неявно используется информация из вполне определенных сегментных регистров. В зависимости от режима работы процессора по их содержимому определяются адреса памяти, с которых начинаются соответствующие сегменты. В программной модели IA-32 имеется шесть *сегментных регистров* CS, SS, DS, ES, GS, FS, служащих для доступа к четырем типам сегментов.

Сегмент кода содержит команды программы. Для доступа к этому сегменту служит *регистр сегмента кода* (Code Segment register) CS. Он содержит адрес сегмента с машинными командами, к которому имеет доступ процессор (эти команды загружаются в конвейер процессора).

Сегмент данных содержит обрабатываемые программой данные. Для доступа к этому сегменту служит *регистр сегмента данных* (Data Segment register) DS, который хранит адрес сегмента данных текущей программы.

Сегмент стека представляет собой область памяти, называемую стеком. Работу со стеком процессор организует по следующему принципу: последний записанный в эту область элемент выбирается первым. Для доступа к этой области служит *регистр сегмента стека* (Stack Segment register) SS, содержащий адрес сегмента стека.

Дополнительный сегмент данных. Неявно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в регистре сегмента данных DS. Если программе недостаточно одного сегмента данных, то она имеет возможность задействовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в регистре DS, при использовании дополнительных сегментов

данных их адреса требуется указывать явно с помощью специальных префиксов переопределения сегментов в команде. Адреса дополнительных сегментов данных должны содержаться в *регистрах дополнительного сегмента данных* (Extension Data Segment registers) ES, GS, FS.

Регистры состояния и управления. В процессор включены два регистра, постоянно содержащие информацию о состоянии как самого процессора, так и программы, команды которой он в данный момент обрабатывает:

- регистр-указатель команд EIP/IP;
- регистр флагов EFLAGS/FLAGS.

С помощью этих регистров можно также ограниченным образом управлять состоянием процессора.

Регистр-указатель команд (Instruction Pointer register) EIP/IP имеет разрядность 32(16) бита и содержит смещение следующей подлежащей выполнению команды относительно содержимого регистра сегмента кода CS в текущем сегменте команд. Этот регистр непосредственно недоступен программисту, но загрузка и изменение его значения производятся различными командами управления, к которым относятся команды условных и безусловных переходов, вызова процедур и возврата из процедур. Возникновение прерываний также приводит к модификации регистра EIP/IP.

Разрядность *регистра флагов* (flag register) EFLAGS/FLAGS равна 32(16) битам. Отдельные биты данного регистра имеют определенное функциональное назначение и называются *флагами*. Младшая часть регистра EFLAGS/FLAGS полностью аналогична регистру FLAGS процессора i8086. На рисунке 2 показано содержимое регистра EFLAGS.

Исходя из особенностей использования, флаги регистра EFLAGS/FLAGS можно разделить на три группы.

В первую группу флагов регистра EFLAGS/FLAGS входят 8 *флагов состояния*. Эти флаги могут изменяться после выполнения машинных команд. Флаги состояния регистра EFLAGS отражают особенности результата исполнения арифметических или логических операций. Это дает возможность анализировать состояние вычислительного процесса и реагировать на него с помощью команд условных переходов и вызовов подпрограмм.

Флаг переноса (carry flag) CF: 1 — арифметическая операция произвела перенос из старшего бита результата, старшим является 7-й, 15-й или 31-й бит в зависимости от размерности операнда; 0 — переноса не было.

:

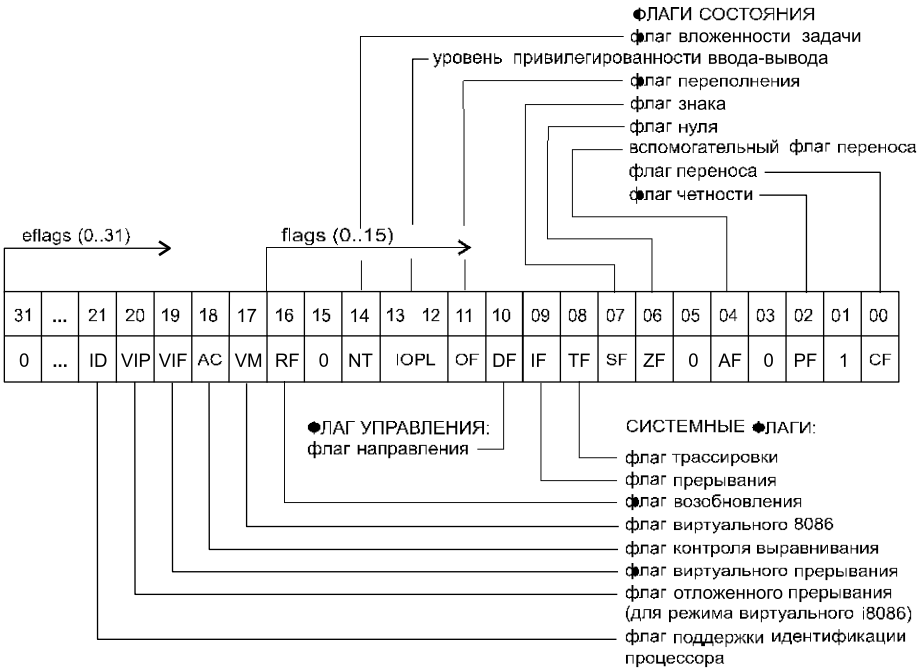


Рисунок 2 – Содержимое регистра eflags

Флаг четности (parity flag) PF: 1 — 8 младших разрядов (этот флаг только для 8 младших разрядов операнда любого размера) результата содержат четное число единиц; 0 — 8 младших разрядов результата содержат нечетное число единиц.

Вспомогательный флаг переноса (auxiliary carry flag) AF применяется только для команд, работающих с BCD-числами. Фиксирует факт заема из младшей тетрады результата: 1 — в результате операции сложения был произведен перенос из разряда 3 в старший разряд или при вычитании был заем в разряд 3 младшей тетрады из значения в старшей тетраде; 0 — переносов и заемов в третий разряд (из третьего разряда) младшей тет-рады результата не было.

Флаг нуля (zero flag) ZF: 1 — результат нулевой; 0 — результат ненулевой.

Флаг знака (sign flag) SF отражает состояние старшего бита результата (биты 7, 15 или 31 для 8-, 16- или 32-разрядных операндов соответственно): 1 — старший бит результата равен 1; 0 — старший бит результата равен 0.

Флаг переполнения (overflow flag) OF используется для фиксации факта потери значащего бита при арифметических операциях: 1 — в результате операции происходит перенос в старший знаковый бит результата или заем из старшего знакового бита результата (биты 7, 15 или 31 для 8-, 16- или 32-разрядных операндов соответственно); 0 — в результате операции не происходит переноса в старший знаковый бит результата или заема из старшего знакового бита результата.

Уровень привилегированности ввода-вывода (Input/Output privilege

level) IOPL используется в защищенном режиме работы процессора для контроля доступа к командам ввода-вывода в зависимости от привилегированности задачи.

Флаг вложенности задачи (nested task) NT используется в защищенном режиме работы процессора для фиксации того факта, что одна задача вложена в другую.

Во вторую группу флагов (группа флагов управления) регистра EFLAGS/FLAGS входит всего один *флаг направления* (directory flag) DF. Он находится в десятом бите регистра EFLAGS и используется цепочечными командами. Значение флага DF определяет направление поэлементной обработки в этих операциях: от начала строки к концу (DF = 0) либо, наоборот, от конца строки к ее началу (DF = 1). Для работы с флагом DF существуют специальные команды CLD (снять флаг DF) и STD (установить флаг DF). Применение этих команд позволяет привести флаг DF в соответствие с алгоритмом и обеспечить автоматическое увеличение или уменьше-ние счетчиков при выполнении операций со строками.

В третью группу флагов регистра EFLAGS/FLAGS входит 8 *системных флагов*, управляющих вводом-выводом, маскируемыми прерываниями, отладкой, переключением между задачами и режимом виртуального процессора 8086. Прикладным программам не рекомендуется модифицировать без необходимости эти флаги, так как в большинстве случаев это ведет к прерыванию работы программы. Далее перечислены системные флаги и их назначение.

Флаг трассировки (trace flag) TF предназначен для организации пошаговой работы процессора.

Флаг прерывания (interrupt enable flag) IF предназначен для разрешения или запрещения (маскирования) аппаратных прерываний (прерываний по входу INTR).

Флаг возобновления (resume flag) RF используется при обработке прерываний от регистров отладки.

Флаг режима виртуального процессора 8086 (virtual 8086 mode) VM является признаком работы процессора в режиме виртуального 8086.

Флаг контроля выравнивания (alignment check) AC предназначен для разрешения контроля выравнивания при обращениях к памяти.

Флаг виртуального прерывания (virtual interrupt flag) VIF.

Флаг отложенного виртуального прерывания (virtual interrupt pending flag) VIP.

Флаг идентификации (identification flag) ID используется для того, чтобы показать факт поддержки процессором инструкции CPUID. Если программа может установить или сбросить этот флаг, это означает, что данная модель процессора поддерживает инструкцию CPUID.

2 Организация памяти

Физическая память, к которой процессор имеет доступ по шине адреса (см. рис. 1.1), называется *оперативной памятью* (или оперативным запоминающим устройством — ОЗУ). На самом нижнем уровне память компьютера можно рассматривать как массив *битов*. Один бит может хранить значение 0 или 1. Для физической реализации битов и работы с ними идеально подходят логические схемы. Но процессору неудобно работать с памятью на уровне битов, поэтому реально ОЗУ организовано как последовательность ячеек — *байтов*. Один байт состоит из восьми битов. Каждому байту соответствует свой уникальный адрес (его номер), называемый *физическим*. Диапазон значений физических адресов зависит от разрядности шины адреса процессора. Для i486 и Pentium он находится в пределах от 0 до $2^{32} - 1$ (4 Гбайт). Для процессоров Pentium Pro/II/III/IV этот диапазон шире — от 0 до 236 – 1 (64 Гбайт).

Механизм управления памятью полностью аппаратный. Это означает, что программа не может сама сформировать физический адрес памяти на адресной шине.

Процессор аппаратно поддерживает две модели использования оперативной памяти.

В *сегментированной модели* программе выделяются непрерывные области памяти (сегменты), а сама программа может обращаться только к данным, которые находятся в этих сегментах.

Страничную модель можно рассматривать как надстройку над сегментированной моделью. В случае использования этой модели оперативная память рассматривается как совокупность блоков фиксированного размера (4 Кбайт и более). Основное применение этой модели связано с организацией виртуальной памяти, что позволяет операционной системе использовать для работы программ пространство памяти большее, чем объем физической памяти. Для процессоров i486 и Pentium размер возможной виртуальной памяти может достигать 4 Тбайт.

Сегментация — механизм адресации, обеспечивающий существование нескольких независимых адресных пространств как в пределах одной задачи, так и в системе в целом для защиты задач от взаимного влияния. В основе механизма сегментации лежит понятие *сегмента*, который представляет собой независимый поддерживаемый на аппаратном уровне блок памяти.

Сегменты начинаются по адресам, кратным 16, то есть имеющим четыре нулевых младших бита. В любой момент времени программе доступны 4 сегмента: сегмент кода, сегмент стека, сегмент данных, дополнительный сегмент. Каждый сегмент определяется путем размещения старших 16 бит адреса первого байта сегмента в одном из четырех сегментных регистров. Таким образом, сегментный регистр содержит адрес начала сегмента.

Обращение к байтам или словам внутри сегмента осуществляется с помощью 16-битного внутрисегментного смещения. Микропроцессор образует 20-битовый адрес байта (слова), суммируя 16-битное смещение с содержимым 16-битного сегментного регистра, к которому добавляются четыре нуля:

Физический адрес = смещение + 16·(сегментный регистр).

Умножение на 16 соответствует добавлению к содержимому сегмента 4 младших нулевых битов.

3 Нотация языка Assembler

Любая команда языка ассемблер состоит из полей метки, мнемокода, операнда и комментария.

Формат команды:

```
[метка:] код [операнд] [;комментарий]
```

Обязательным в записи команды ассемблера является только поле мнемокода. Поля команды должны быть разделены хотя бы одним пробелом.

Метка может содержать до 31 символа и должна заканчиваться двоеточием. В имя метки могут входить:

– буквы от A до Z или от a до z;

– цифры от 0 до 9;

– специальные знаки: ? . @ _ $

Метку можно начать любым символом, кроме цифры. Если используется точка, то она должна быть только первым символом метки.

Примеры меток:

```
Metka
.cikl Get_cikl
```

Поле операнда может содержать источник и приемник, причем ***приемник указывается первым***. Источник от приемника отделяются запятой:

```
Приемник, Источник
```

В ассемблере используются следующие константы:

1) двоичная – число, записанное в двоичной системе счисления, заканчивающееся буквой B, например: `1001010B`;

2) десятичная – число, записанное в десятичной системе счисления, заканчивающееся буквой D, например: `1234D` или `1234`;

3) шестнадцатеричная – число, записанное в шестнадцатеричной системе счисления, заканчивающееся буквой H, например: `67ADH`;

4) литерал – строка букв, цифр и других символов, заключенная в апострофы или кавычки, например: `'Ассемблер'`.

В качестве констант можно использовать и отрицательные числа. В

случае десятичного числа перед ним достаточно поставить знак минус. Если число двоичное или шестнадцатеричное, то его необходимо записать в дополнительном коде.

Комментарии используются для пояснения текста программы. Для задания многострочных комментариев нужно в начале каждой строки поставить точку с запятой.

Псевдооператоры. Псевдооператоры управляют работой ассемблера, а не микропроцессора. В отличие от команд ассемблера большинство псевдооператоров не генерируют объектного кода и выполняются на шаге трансляции программы.

Формат задания псевдооператора: `[идентификатор] псевдооператор [операнд] [;комменарий]`

Обязательным является только поле псевдооператора.

Псевдооператоры определения имен. Эти псевдооператоры позволяют программисту определить символические имена для часто используемых выражений. Такие имена создаются с помощью псевдооператоров EQU и =. После присваивания выражению имени можно использовать это имя всюду, где требуется указать выражение. Определенные псевдооператором = имена можно переопределить, а определенные псевдооператором EQU нельзя. Псев- дооператор EQU можно использовать как с числовыми, так и с тексто- выми выражениями, а псевдооператор = только с числовыми.

Пример:
```
count equ cx
const = 512
const = 2*const+1
```

Псевдооператоры определения переменных. Поименованные элементы данных, содержимое которых может быть изменено, называются переменными. Любая переменная имеет три атрибута: сегмент, смещение и тип.

Сегмент (SEG) определяет сегмент, содержащий переменную.

Смещение (OFFSET) представляет собой расстояние или дистанцию в байтах от начала сегмента.

Тип (TYPE) определяет единицу памяти, выделяемую для хранения переменной.

Псевдооператоры определения переменных:
```
[имя] db выражение [,…]
[имя] dw выражение [,…]
[имя] dd выражение [,…]
```

Псевдооператор db резервирует байты, dw – слова, dd – двойныеслова. Выражение может быть числовым или адресным.

Пример:

```
a db 4
b dw 6
c db 3*6-2     ;числовое выражение
d db a         ;адресное выражение
```

При определении переменной без присваивания ей начального значения необходимо указать в поле выражения вопросительный знак `c db ?`

Если имя переменной не указывается, то просто резервируется память:
`db ?`

Простые типы данных:

`db` – резервирование памяти размером в 1 байт;

`dw` – резервирование памяти размером в 2 байт;

`dd` – резервирование памяти размером в 4 байт;

`df` – резервирование памяти размером в 6 байт;

`dp` – резервирование памяти размером в 6 байт;

`dq` – резервирование памяти размером в 8 байт;

`dt` – резервирование памяти размером в 10 байт.

Псевдооператоры определения сегментов. В исходной программе должны быть определены сегменты, из которых состоит программа. Может быть определено до четырех сегментов: сегмент кода, сегмент данных, сегмент стека и дополнительный сегмент.

Формат описания сегмента:

```
Имя segment [тип подгонки] [тип связи] ['класс']
…
Имя ends
```

Существуют два типа подгонки или выравнивания сегмента:

`para` – сегмент будет расположен начиная с ячейки памяти, адрес которой кратен 16. Это значит, что первый байт сегмента будет иметь нулевое смещение относительно сегментного регистра;

`byte` – располагает сегмент начиная с любого адреса.

Тип связи определяет способ, которым сегмент объединяется с другими сегментами:

`public` – вызывает объединение всех сегментов с одним и тем же именем в виде одного большого сегмента;

`at` – располагает сегмент по заданному абсолютному адресу. Класс определяет категорию сегмента:

`code` – сегмент кода;

`data` – сегмент данных;

`extra` – дополнительный сегмент;

`stack` – сегмент стека.

Псевдооператор `segment` не сообщает ассемблеру, какого рода сегмент

должен быть определен. Для этой цели служит псевдооператорassume.

Формат оператора:

```
assume сегментный_регистр:имя_сегмента[,…]
```

Оператор assume связывает каждый сегментный регистр с сегментом, который сегментный регистр адресует в данный момент.

Командой assume нельзя загрузить адрес начала сегмента в соответствующий сегментный регистр. Это осуществляется через регистр общего назначения.

Оператор assume помещается сразу же за оператором segment, определяющим сегмент кода:

```
cseg segment para public 'code'
assume ds:dseg, cs:cseg, ss:sseg
mov ax,dseg
mov ds,ax
...
cseg ends
```

Псевдооператоры определения процедур. *Формат описания процедуры*:

```
имя_процедуры proc атрибут_дистанции
...
ret
имя_процедуры endp
```

Каждая процедура должна начинаться оператором proc и заканчиваться оператором endp. Процедура должна содержать команду возврата из процедуры ret.

Атрибуты дистанции:

far – дальняя процедура,

near – близкая процедура.

Процедура с атрибутом near может быть вызвана только из того сегмента команд, в котором она определена.

Формат вызова процедуры:

```
call имя_процедуры
```

Псевдооператоры управления трансляцией. Псевдооператор end отмечает конец исходной программы и указывает ассемблеру, где завершить трансляцию. Псевдооператор end должен присутствовать в каждой программе.

Формат оператора:

```
end [метка точки входа]
```

где метка определяет исходную программу. Например, оператор

```
end proc
```

отмечает конец программы proc.

Команды пересылки. *Формат команды:*

```
mov приемник, источник ;И→П
```

Команда позволяет пересылать байт или слово между регистром и ячейкой памяти или между двумя регистрами:

```
mov регистр1, регистр2mov регистр,
память
```

В команде `mov` запрещается пересылка из одной ячейки памяти в другую.

```
mov память1, память2 ;НЕЛЬЗЯ!
```

Нельзя загрузить непосредственно адресуемый операнд в регистр сегмента. Это можно сделать только через регистр общего назначения

```
mov ds,dseg ;НЕЛЬЗЯ!

mov ax,dseg ;НУЖНО!
mov ds,ax
```

Нельзя пересылать значение одного регистра сегмента в другой регистр сегмента. Подобная пересылка делается через регистр общего назначения.

```
mov ds,es ;НЕЛЬЗЯ!

mov ax,es ;НУЖНО!mov ds,ax
```

Нельзя использовать регистр CS в качестве приемника в команде пересылки.

Для работы со стеком используются следующие команды:

```
push источник ;поместить слово в стек
pop приемник  ;извлечь слово из стека
```

4 Арифметические команды

Команды сложения. *Формат команды*:

```
add приемник, источник ;П+И→П
```

Команда `add` используется для сложения приемника и источника, результат помещается в приемник.

Оба операнда не могут быть именами переменных. Источникомможет быть число.

Можно складывать:
ПАМЯТЬ+РЕГИСТР
РЕГИСТР+ПАМЯТЬ
РЕГИСТР +РЕГИСТР
ПАМЯТЬ+ЧИСЛО
РЕГИСТР+ЧИСЛО
Нельзя складывать
ПАМЯТЬ + ПАМЯТЬ

```
inc приемник ;П+1→П
```

Команда inc используется для увеличения содержимого прием-ника на единицу. Приемник – это регистр или ячейка памяти.

Команды вычитания. *Формат команды*:

```
sub приемник, источник ;П-И→П
```

Команда sub используется для вычитания содержимого источника из содержимого приемника, результат помещается в приемник.

Оба операнда не могут быть именами переменных. Источником может быть число.

Можно вычитать:
ПАМЯТЬ – РЕГИСТР
РЕГИСТР – ПАМЯТЬ
РЕГИСТР – РЕГИСТР
ПАМЯТЬ – ЧИСЛО
РЕГИСТР – ЧИСЛО
Нельзя вычитать
ПАМЯТЬ – ПАМЯТЬ

```
dec приемник ;П-1→П
```

Команда dec используется для уменьшения содержимого прием-ника на единицу. Приемник – это регистр или ячейка памяти.

Команды умножения. *Формат команды умножения чисел без знака*:

```
mul источник  ;al*И→ax  при работе с байтами
;ax*И→ax-dx       при работе со словами
```

Формат команды умножения чисел со знаком:

```
imul источник ;al*И→ax    при работе с байтами
;ax*И→ax-dx       при работе со словами
```

Команды mul и imul умножают содержимое регистра al или ax (в зависимости от размерности операндов) на содержимое источника, указанного в команде умножения. Источник – это или регистр или ячейка памяти. В результате работы команд умножения с данными длиной байт регистр al расширяется до ax, а с данными длиной слово

- регистр ax расширяется до dx.

***НЕЛЬЗЯ использовать в команде умножения в качества источника непосредственное значение*!**

Команды деления. *Формат команды деления чисел без знака*:

```
div источник  ;al/И→al-ah при работе с байтами
;ax/И→ax-dx       при работе со словами
;al – частное, ah – остаток
;ax – частное, dx – остаток
```

Формат команды деления чисел со знаком:

```
idiv источник ;al/И→al-ah при работе с байтами
;ax/И→ax-dx       при работе со словами
;al – частное, ah – остаток
;ax – частное, dx – остаток
```

Команды `div` и `idiv` делят содержимое регистра `al` или `ax` (в зависимости от размерности операндов) на содержимое источника, указанного в команде умножения. Источник – это или регистр или ячейка памяти. В результате работы команд умножения с данными длиной байт в регистр `al` помещается частное, в регистр `ah` – остаток, а с данными длиной слово – в регистр `ax` помещается частное, в регистр `dx` – остаток.

***НЕЛЬЗЯ использовать в команде деления в качества источника непосредственное значение*!**

Команды расширения знака. *Форматы команд*:

```
cbw ;расширить байт до слова
cwd ;расширить слово до двойного слова
```

Команда `cbw` воспроизводит 7 бит регистра AL во всех битах ре- гистра AH.

Команда `cwd` воспроизводит 15 бит регистра AX во всех битах регистра DX.

5 Пример стандартной программы на ассемблере

Ниже приведен листинг с программой на ассемблере, выводящую на консоль сообщение: «Hello World! No war and bomb! Let's live friendly and learn assembler language».

```
data    segment para public 'data'
message db      'Hello World! No war and bomb! Let us live friendly and learn assembler
                language. $'
data    ends
stk     segment stack
        db 256 dup ('?'); сегмент стека
stk ends
code
segment para public 'code'      ; начало сегмента кода
main    proc    ; начало процедуры main
        assume cs:code,ds:data,ss:stk
        mov ax,data     ; адрес сегмента данных в регистр ax mov ds,ax    ; ax в ds
        mov ah,9
        mov dx,offset message
        int 21h  ; вывод сообщения на экран
        mov ax,4c00h  ; пересылка 4c00h в регистр ax
```

```
        int 21h  ; вызов прерывания с номером 21h
main    endp     ; конец процедуры main
code    ends     ; конец сегмента кода
end     main     ; конец программы с точкой входа main
```

6 Пример функции с ассемблерной вставкой (C++ Visual Studio)

```
int func()
 {
  __asm
    {
     …
     mov eax, 1
     pop cx
     …
     }
 return 0;
 }
```

7 Программирование ветвящихся вычислительных процессов

Для программирования ветвящихся вычислительных процессов, используются команда вычитания cmp и команды передачи управления.

Команда вычитания. Формат команды вычитания:

cmp <приемник>, <источник>

Эта команда вычитает операнд-источник из операнда-приемника, но не сохраняет результат вычитания в операнде-приемнике, а только соответствующим образом воздействует на флаги.

Команды передачи управления. Команды передачи управления делятся на 4 группы:

1. Команды безусловной передачи управления.
2. Команды условной передачи управления.
3. Команды управления циклами.
4. Команды работы с процедурами.

Команды условной передачи управления. Команды условной передачи управления позволяют принять решение в зависимости от определенного условия. Если условие истинно, то осуществляется переход по указанной в команде метке. В противном случае выполняется команда, следующая за командой перехода.

Обычно команды условной передачи управления используются совместно с командой сравнения cmp.

Формат команды условного перехода:

jx <метка>,

где x – модификатор команды;

<метка> – метка перехода, которая находится не далее –128 или +127 байтов от команды условной передачи.

В таблице 1 представлены некоторые команды условного перехода.

Таблица 1 – Команды условного перехода

Условие перехода	Следующая за cmp команда	
	для чисел без знака	для чисел со знаком
приемник > источник	ja	jg
приемник = источник	je	je
приемник < источник	jb	jl
приемник ≥ источник	jae	jge
приемник ≤ источник	jbe	jle
приемник ≠ источник	jne	jne

Пример:

```
        …
        cmp ax, bx   ;сравниваем содержимое регистров ax и bx
        jg met1      ;если ax>bx то переход на met1
        jl met2      ;если ax<bx то переход на met2
        je met3      ;если ax=bx то переход на met3
        …
met1:
        …
met2:
        …
met3:
        …
```

Команда безусловного перехода. Эта команда используется для обхода группы команд, которым передается управление из другой части программы.

Формат команды:

jmp <метка>,

где метка – имя метки перехода, которая находится не далее –128 или +127 байтов от команды безусловного перехода.

Псевдооператоры определения процедур. Формат описания процедуры:

```
<имя_процедуры> proc <атрибут_дистанции>
...
ret
<имя_процедуры> endp
```

Каждая процедура должна начинаться оператором proc и заканчиваться оператором endp. Процедура должна содержать команду возврата из процедуры ret.

Атрибуты дистанции:

- far – дальняя процедура;
- near – близкая процедура.

Процедура с атрибутом near может быть вызвана только из того сегмента команд, в котором она определена.

Формат вызова процедуры:

```
call <имя_процедуры>
```

8 Программирование циклических вычислительных процессов

Команды управления циклами.

Команда loop.

Эта команда уменьшает содержимое регистра cx на 1 и передает управление оператору, помеченному меткой, если содержимое регистра cx≠0. Завершение выполнения цикла происходит в том случае, если содержимое регистра cx уменьшается до нуля.

Формат команды:

```
loop <метка>
```

Пример:

```
        …
        mov cx,10
 met1:
        …
        ;тело цикла
        loop met1
```

Команда loope (loopz).

Эта команда уменьшает содержимое регистра cx на 1, а затем осуществляет переход, если содержимое регистра cx≠0 или флаг нуля zf=1. Повторение цикла завершается, если либо содержимое регистра cx=0, либо флаг zf=0, либо оба они равны 0. Команда loope обычно используется для

поиска первого ненулевого результата в серии операций. Синонимом команды loope является команда loopz.

Команда loopne (loopnz).

Эта команда уменьшает содержимое регистра cx на 1, а затем осуществляет переход, если содержимое регистра cx≠0 и флаг нуля zf=0. Повторение цикла завершается, если либо содержимое регистра cx=0, либо флаг zf=1, либо будет выполнено и то и другое. Команда loopne обычно используется для поиска первого нулевого результата в серии операций. Синонимом команды loopne является команда loopnz.

9 Режимы адресации

Под режимами адресации понимаются способы доступа к данным.

Все режимы адресации можно условно разделить на 7 групп.

Для доступа к операнду используется 20-битовый физический адрес.

Физический адрес операнда получается сложением значения смещения адреса операнда с содержимым сегментного регистра, предварительно дополненного четырьмя нулями.

Смещение адреса операнда называется исполнительным адресом. Исполнительный адрес показывает, на каком расстоянии в байтах располагается операнд от начала сегмента, в котором он находится.

Будучи 16-битовым числом без знака, исполнительный адрес позволяет получить доступ к операндам, находящимся выше начала сегмента на расстоянии 64 Кбайт.

При работе с адресацией операндов необходимо помнить, что микропроцессор хранит 16-битовые числа в обратном порядке, а именно: младшие биты числа в байте с меньшим адресом.

Регистровая. Этот режим предполагает, что микропроцессор извлекает операнд из регистра или загружает его в регистр.

```
mov cx, ax
inc di
```

Непосредственная. Этот режим адресации позволяет указывать 8- или 16-битовые значения константы в операнде-источнике.

```
mov cx, 100
mov al, -10
```

Непосредственный операнд может быть идентификатором, определенным операторами equ или =

```
k equ 100
…
mov cx,k
```

Прямая. При прямой адресации исполнительный адрес является составной частью команды. Микропроцессор добавляет этот исполнительный адрес к сдвинутому содержимому регистра данных ds и получает 20-битовый физический адрес. По этому адресу и находится операнд. Обычно прямая адресация применяется, если операндом служит метка переменной.

```
mov ax, table
```

Косвенная регистровая адресация. В этом случае исполнительный адрес операнда содержится в базовом регистре bx, регистре указателя базы bp, регистре sp или индексных регистрах si и di.

```
mov ax, [bx]
```

Смещение адреса в регистр можно поместить оператором offset или lea

```
mov bx, offset table
lea bx, table
```

Адресация по базе. Ассемблер вычисляет исполнительный адрес с помощью сложения значения сдвига с содержимым регистров bx или bp. Адресация по базе используется при доступе к структурированным записям данных, расположенных в разных участках памяти. В этом случае базовый адрес записи помещается в базовый регистр bx или bp и доступ к отдельным его элементам записи осуществляется по сдвигу относительно базы. А для доступа к разным записям одной и той же структуры достаточно соответствующим образом изменить содержимое базового регистра.

```
mas db 1,2,3,4,5,6,7,8,9,10
mov bx, offset mas
mov al,[bx+4] ;загрузка в al mas[4]=5
```

Прямая адресация с индексированием. Исполнительный адрес вычисляется как сумма значений смещения и индексного регистра di или si. Этот тип адресации удобен для доступа к элементам таблицы, когда смещение указывает на начало таблицы, а индексный регистр на номер элемента.

```
table db 10 dup (?)
mov di, 5
mov al, table[di] ;загрузка 6 элемента таблицы
```

Адресация по базе с индексированием. Исполнительный адрес вычисляется как сумма значений базового регистра, индексного регистра и сдвига. Этот режим адресации удобен при адресации двумерного массива. Пусть задан двумерный массив A:

$$A=\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

```
;выделение памяти под элементы матрицы
a db 1,2,3,4,5,6,7,8,9,10,11,12
;доступ к элементу a[1,2]=7 двумерного массива
mov bx, offset a
mov di,4
mov al, [bx][di+2]
```

Допускаются следующие формы записи адресации по базе с индексированием:

```
mov al, [bx+2+di]
mov al, [di+bx+2]
mov al, [bx+2][di]
```

Из семи режимов адресации самыми быстрыми являются регистровая и непосредственная адресации операндов. В других режимах адресация выполняется дольше, так как вначале необходимо вычислить адрес ячейки памяти, извлечь операнд, а затем передать его операционному блоку.

10 Цепочечные команды

Цепочечные команды также называют командами обработки строк символов. Под строкой символов понимается последовательность байт, а цепочка – это более общее название для случаев, когда элементы последовательности имеют размер слово или двойное слово. То есть цепочечные команды позволяют проводить действия над блоками памяти, представляющими собой последовательности элементов следующего размера: 8 бит (байт); 16 бит (слово); 32 бита (двойное слово). Содержимое этих блоков для микропроцессора не имеет никакого значения. Это могут быть символы, числа и все что угодно. Главное, чтобы размерность элементов совпадала с одной из вышеперечисленных, и эти элементы

находились в соседних ячейках памяти.

В системе команд микропроцессора имеются семь операций-примитивов обработки цепочек. Каждая из них реализуется в микропроцессоре тремя командами, в свою очередь, каждая из этих команд работает с соответствующим размером элемента – байтом, словом или двойным словом. Особенность всех цепочечных команд в том, что они, кроме обработки текущего элемента цепочки, осуществляют еще и автоматическое продвижение к следующему элементу данной цепочки.

Операции-примитивы и команды, с помощью которых они реализуются:

1. Пересылка цепочки:

```
movs <адрес_приемника>, <адрес_источника>
                ;(MOVe String) – переслать цепочку;
movsb           ;(MOVe String Byte) – переслать цепочку байтов;
movsw ;(MOVe String Word) – переслать цепочку слов;
movsd           ;(MOVe String Double word) – переслать цепочку двойных
                ;слов.
```

Команды производят копирование элементов из одной области памяти (цепочки) в другую.

2. Сравнение цепочек:

```
cmps <адрес_приемника>, <адрес_источника>
                ;(CoMPare String) – сравнить строки;
cmpsb           ;(CoMPare String Byte) – сравнить строку байт;
cmpsw ;(CoMPare String Word) – сравнить строку слов;
cmpsd           ;(CeMPare String Double word) – сравнить строку двойных
                ;слов.
```

Команды производят сравнение элементов цепочки-источника с элементами цепочки-приемника.

3. Сканирование цепочки:

```
scas <адрес_приемника> ;(SCAning String) – сканировать цепочку;
scasb           ;(SCAning String Byte) – сканировать цепочку байт;
scasw           ;(SCAning String Word) – сканировать цепочку слов;
scasd           ;(SCAning String Double Word) – сканировать цепочку
                ;двойных слов.
```

Команды производят поиск некоторого значения в области памяти.

4. Загрузка элемента из цепочки:

```
lods <адрес_источника> ;(LOaD String) – загрузить элемент из цепочки
```

```
                ;в регистр-аккумулятор al/ax/eax;
lodsb           ;(LOaD String Byte) – загрузить байт из цепочки в регистр al;
lodsw           ;(LOaD String Word) – загрузить слово из цепочки в регистр
                ;ax;
lodsd           ;(LOaD String Double Word) – загрузить двойное слово
                ;из цепочки в регистр eax.
```

Эта операция позволяет извлечь элемент цепочки и поместить его в регистр-аккумулятор al, ax или eax.

5. Сохранение элемента в цепочке:

```
stos <адрес_приемника> ;(STOre String) – сохранить элемент из
                ;регистра-аккумулятора al/ax/eax в цепочке;
stosb           ;(STOre String Byte) – сохранить байт из регистра al в
                ;цепочке;
stosw           ;(STOre String Word) – сохранить слово из регистра ax
                ;в цепочке;
stosd           ;(STOre String Double Word) – сохранить двойное слово из
                ;регистра eax в цепочке.
```

Эта операция позволяет произвести действие, обратное команде lods, то есть сохранить значение из регистра-аккумулятора в элементе цепочки.

6. Получение элементов цепочки из порта ввода-вывода:

```
ins <адрес_приемника>, <номер_порта>
                ;(INput String) – ввести элементы из
                ;порта ввода-вывода в цепочку;
insb            ;(INput String Byte) – ввести из порта цепочку байтов;
insw            ;(INput String Word) – ввести из порта цепочку слов;
insd            ;(INput String Double Word) – ввести из порта цепочку
                ;двойных слов.
```

7. Вывод элементов цепочки в порт ввода-вывода:

```
outs <номер_порта>, <адрес_источника>
                ;(OUTput String) – вывести
                ; элементы из цепочки в порт ввода-вывода;
outsb           ;(OUTput String Byte) – вывести цепочку байтов в порт
                ;ввода-вывода;
outsw           ;(OUTtput String Word) – вывести цепочку слов в порт
                ;ввода-вывода;
outsd           ;(OUTput String Double Word) – вывести цепочку двойных
                ;слов в порт ввода-вывода.
```

К цепочечным командам нужно отнести и *префиксы повторения:*

rep
repe или repz
repne или repnz

Префиксы повторения указываются перед нужной цепочечной командой в поле метки. Цепочечная команда без префикса выполняется один раз. Размещение префикса перед цепочечной командой заставляет ее выполняться в цикле. Отличия префиксов в том, на каком основании принимается решение о циклическом выполнении цепочечной команды: по состоянию регистра ecx/cx или по флагу нуля zf:

- префикс повторения rep (REPeat) используется с командами, реализующими операции-примитивы пересылки и сохранения элементов цепочек movs и stos. Префикс rep заставляет данные команды выполняться, пока содержимое в ecx/cx не станет равным 0. При этом цепочечная команда, перед которой стоит префикс, автоматически уменьшает содержимое ecx/cx на единицу. Та же команда, но без префикса, этого не делает;

- префиксы повторения repe или repz (REPeat while Equal or Zero) являются синонимами. Цепочечная команда выполняется до тех пор, пока содержимое ecx/cx не равно нулю или флаг zf равен 1. Как только одно из этих условий нарушается, управление передается следующей команде программы. Благодаря возможности анализа флага zf наиболее эффективно эти префиксы можно использовать с командами cmps и scas для поиска отличающихся элементов цепочек;

- префиксы повторения repne или repnz (REPeat while Not Equal or Zero) также являются синонимами. Префиксы repne/repnz заставляют цепочечную команду циклически выполняться до тех пор, пока содержимое ecx/cx не равно нулю или флаг zf равен нулю. При невыполнении одного из этих условий работа команды прекращается. Данные префиксы также можно использовать с командами cmps и scas, но для поиска совпадающих элементов цепочек.

Особенность формирования физического адреса операндов адрес_источника и адрес_приемника. Цепочка-источник, адресуемая операндом адрес_источника, может находиться в текущем сегменте данных, определяемом регистром ds. Цепочка-приемник, адресуемая операндом адрес_приемника, должна быть в дополнительном сегменте данных, адресуемом сегментным регистром es. Допускается замена (с помощью префикса замены сегмента) только регистра ds, регистр es заменять нельзя. Вторые части адресов – смещения цепочек должны находиться:

- для цепочки-источника это регистр esi/si;
- для цепочки-получателя это регистр edi/di.

Таким образом, полные физические адреса для операндов

цепочечных команд следующие:

- адрес_источника – пара ds:esi/si;
- адрес_приемника – пара es:edi/di.

Есть две возможности задания направления обработки цепочки: от начала цепочки к её концу, то сеть в направлении возрастания адресов; от конца цепочки к началу, то есть в направлении убывания адресов.

Направление определяется значением флага направления df (Direction Flag) в регистре eflags/flags:

- если df=0, то значения индексных регистров esi/si и edi/di будут автоматически увеличиваться (операция инкремента) цепочечными командами, то есть обработка будет осуществляться в направлении возрастания адресов;
- если df=1, то значения индексных регистров esi/si и edi/di будут автоматически уменьшаться (операция декремента) цепочечными командами, то есть обработка будет идти в направлении убывания адресов.

Состоянием флага df можно управлять с помощью двух команд, не имеющих операндов:

- cld (Clear Direction Flag) – очистить флаг направления. Команда сбрасывает флаг направления df в 0.
- std (Set Direction Flag) – установить флаг направления. Команда устанавливает флаг направления df в 1.

11 Лабораторная работа №1. Изучение аппаратно-программной архитектуры процессоров семейства Intel – разработка ассемблерной вставки

Цель работы: изучение программной модели микропроцессора, регистров, организации памяти; практическая работа с ассемблерными вставками

1. Изучить следующие теоретические сведения: программная модель микропроцессора; регистры и их назначение; организация памяти; основные конструкции и базовые команды ассемблера (используя материалы лекций и методические указания).
2. Написать ассемблерную вставку, реализующую обработку строки согласно варианту, выданному преподавателем. Оформить ее в виде отдельной функции.
3. Реализовать данную обработку строки также в виде функции на C++. Сравнить быстродействие обоих вариантов. В отчете отразить выводы.

Содержание отчета:

- титульный лист;
- описание варианта задания;
- текст программы;

- скриншоты результатов.

ПРИМЕЧАНИЕ: для разработки использовать MS Visual Studio.

Варианты заданий

1. Перевернуть строку.
2. Поменять четные символы с нечетными.
3. Даны 4 строки. Поменять 1-ю с 3-ей, 2-ю с 4-й.
4. Даны 2 строки. Совместить четные символы одной строки с нечетными другой.
5. Даны 2 строки. Совместить первую половину строки 1 с первой половиной строки 2.
6. Нечетные символы заменить на символ +.
7. Совместить две строки. Совпадающие символы заменить на 0.
8. Сместить все символы на 1-н вперед циклично.
9. Перевернуть две половины строки.
10. Сместить все символы на 1-н назад циклично.
11. Даны 2 строки. Совместить первую половину строки 1 с второй половиной строки 2.
12. Заменить пробелы на символ табуляции.
13. Четные символы заменить на символ?
14. Удалить повторяющиеся пробелы, также пробелы в начале и в конце строки.
15. Даны 4 строки. Поменять 1-ю с 4-ей, 2-ю с 3-й.
16. Даны 2 строки. Совместить вторую половину строки 1 с второй половиной строки 2.

12 Лабораторная работа №2. Изучение аппаратно-программной архитектуры процессоров семейства Intel – обработка символьных данных

Цель работы: изучение программной модели микропроцессора, практическая работа с подсистемой регистровой и оперативной памяти.

1. *Постановка задачи.* Дан текст – непустая последовательность не длиннее ста символов. Признаком конца ввода является точка, в сам текст точка не входит.

Проверить, удовлетворяет ли текст заданному условию. Если условие выполнено, преобразовать текст по указанному в варианте правилу; в противном случае выдать соответствующее сообщение. Преобразованный текст напечатать.

Проверяемое условие и правила обработки текста определяются конкретным вариантом задания.

Если введенная последовательность символов не является текстом, преобразовывать ее не нужно, а следует напечатать соответствующее сообщение.

Ввод текста, проверка условия, обработка текста и печать результата должны выполняться последовательно, отдельными частями программы.

2. Используя знания и навыки, полученные в результате выполнения лабораторной работы №1, а также изучив дополнительные теоретические материалы, приведенные в методическом указании, написать программу на ассемблере, реализующую обработку текста согласно варианту, выданному преподавателем.

Содержание отчета:

- титульный лист;
- описание варианта задания;
- текст программы;
- скриншоты результатов.

Варианты заданий

Проверяемое условие

1) Текст оканчивается прописной латинской буквой, которая больше в тексте не встречается (с.к.).

2) Текст начинается цифрой и оканчивается цифрой, причем эти цифры различны.

3) Текст начинается и оканчивается латинской буквой.

4) Текст содержит не менее трех латинских букв.

5) Текст содержит равное количество прописных и строчных латинских букв.

6) Текст содержит равное количество прописных латинских и прописных русских букв.

7) Текст не содержит иных символов, кроме цифр и прописных русских букв.

8) В тексте больше латинских букв, чем цифр.

9) За каждой цифрой текста идет строчная латинская буква.

10) Количество строчных латинских букв в тексте равно количеству цифр в нем.

11) В тексте не встречаются одновременно цифра 0 и латинская буква O (в любом регистре).

12) Количество строчных русских букв в тексте равно количеству цифр в нем.

13) В тексте больше латинских букв, чем русских.

14) В текст входят только строчные латинские буквы и цифры, причём буквы и цифры чередуются.

15) Текст начинается и оканчивается русской буквой.

16) Текст содержит не менее пяти русских букв.

Правило преобразования

1) Заменить каждую ненулевую цифру соответствующей ей строчной буквой латинского алфавита (1 → a, 2 → b и т.д.).

2) Заменить каждую строчную латинскую букву цифрой N mod 10, где N – порядковый номер буквы в алфавите.

3) Заменить каждую прописную русскую букву цифрой N mod 10, где N – порядковый номер буквы в алфавите.

4) Заменить каждую прописную латинскую букву следующей за ней по алфавиту, букву Z менять на A.

5) Заменить каждую прописную русскую букву следующей за ней по алфавиту, букву Я менять на А.

6) Заменить каждую строчную латинскую букву соответствующей прописной буквой, а прописную – строчной.

7) Заменить каждую прописную русскую букву соответствующей строчной буквой. 8) Заменить каждую латинскую букву симметричной ей в алфавите (A ↔ Z, b ↔ y, ...).

9) Заменить каждую прописную русскую букву симметричной ей в алфавите (А ↔ Я, Б ↔ Ю, ...).

10) Удвоить каждую литеру текста. (Можно использовать дополнительную память.)

11) Удвоить каждую строчную латинскую букву текста. (Можно использовать дополнительную память.)

12) Циклически сдвинуть текст на три позиции влево. (с.к.)

13) Циклически сдвинуть текст на четыре позиции вправо. (с.к.)

14) Поменять местами каждый символ на нечетной позиции и следующий за ним символ

текста, если таковой имеется.

15) В каждой группе следующих подряд одинаковых литер оставить только одну из них. (с.к.)

16) Удалить из текста все повторные вхождения его первой литеры. (с.к.)

Требования к программе

Нельзя использовать дополнительную память, размер которой зависит от длины текста (в частности, стек) во всех вариантах, кроме тех, где явно не указано обратное.

При реализации вариантов, помеченных "с.к.", требуется использовать строковые команды с префиксами повторения.

Проверку условия и выполнение преобразований текста следует реализовать в виде процедур. Глобальные переменные не использовать.

Интерфейс программы должен быть достаточно удобным: надо печатать приглашение для ввода, пояснять, какое условие проверяется, истинно ли оно и как текст преобразуется.

Рекомендации

В некоторых вариантах удобно реализовать в качестве процедур или макросов проверку, является ли символ буквой, цифрой и т.п.

В процедуре преобразования текста может быть полезным иметь два указателя на позиции в тексте: откуда происходит чтение и куда идёт запись.

Максимально возможную длину текста описать в виде константы.

13 Лабораторная работа №3. Изучение аппаратно-программной архитектуры процессоров семейства Intel – макроопределения

Цель работы: изучение приемов разработки макроопределений использования их в программах.

1. Изучить состав и средства задания макроопределений.
2. Написать макроопределение, реализующее функцию заданного преподавателем варианта работы.
3. Выполнить проверку работоспособности разработанного макроопределения.

Содержание отчета:
- титульный лист;
- описание варианта задания;
- текст программы;
- скриншоты результатов.

Рекомендации

Каждое макроопределение (МО) имеет три части.

Заголовок – псевдооператор MACRO, в поле метки которого указано имя МО, а в поле операнда – необязательный список формальных параметров. В списке формальных параметров указываются переменные – входные параметры, которые могут изменяться при каждом вызове МО.

Тело – последовательность операторов Ассемблера (команд и псевдооператоров), которые задают действия, выполняемые МО.

Концевик – псевдооператор ENDM, который отмечает конец МО.

Синтаксис макроопределения следующий:

```
Имя MACRO [список формальных параметров] ;начало МО
                Тело макроопределения,
                Использующее формальные параметры
ENDM ;ключевое слово – конец МО
```

Имя используется для вызова макроопределения в макрокоманде с фактическими параметрами.

```
Имя [список фактических параметров]
```

Пример:

```
;МО для сложения значений размером в слово
ADD_WORDS MACRO TERM1, TERM2, SUM
                MOV AX, TERM1
                ADD AX, TERM2
                MOV SUM, AX
                ENDM
```

Для Ассемблера безразлично, что будет указано в качестве операндов МО: имена регистров, ячейки памяти или непосредственные значения (конечно, непосредственное значение нельзя использовать в качестве операнда SUM).

Замечание. МО легче передать параметры, чем процедуре: в случае МО набирается имя параметра, в случае процедуры необходимо переслать значение параметра в регистр или ячейку памяти.

Примеры макрокоманд вызова:

```
ADD_WORDS CX, DX, BX ;сложить содержимое регистров CX и DX,
                     ; результат поместить в регистр BX
ADD_WORDS 134, 357, CX ;сложить 134 и 357, результат в регистр CX
ADD_WORDS X, Y, RES   ;сложить значения переменных X и Y, результат
                      ; поместить в переменную RES
```

Макрокоманды используются в тексте программы. Каждая макрокоманда в тексте программы заменяется макрорасширенем.

Варианты заданий

1. Подсчитать количество вхождений заданного символа в строку текста.
2. Заменить заданный символ в строке текста на указанный новый символ.
3. Удалить заданный символ из текста. Подсчитать количество слов в строке, считая словом последовательность знаков между пробелами.
4. Переместить заданный символ, если он содержится в строке, в начало строки.
5. Переместить заданный символ, если он находится в строке, в конец строки.
6. Удалить все пробелы из строки символов.

Задана строка слов. Словом считается последовательность символов, разделенная пробелами. Требуется:

7) в исходной строке оставить между словами лишь по одному пробелу, удалив лишние;

8) в исходной строке изменить порядок следования слов на инверсный;

9) из исходной строки удалить слова, начинающиеся с заданного символа;

10) из исходной строки удалить слово минимального размера;

11) в исходной строке слова, начинающиеся с заданной буквы, заменить знаком $;

12) из исходной строки удалить слова, содержащие хотя бы один русский символ;

13) из исходной строки удалить слова, содержащие хотя бы одну десятичную цифру;

14) из исходной строки удалить слова, содержащие хотя бы один латинский символ

15) в исходной строке слова, заканчивающиеся на заданную букву, заменить знаком &;

16) из исходной строки удалить слово максимального размера.

Список использованных источников

1. Юров В. И. Assembler: Учебник / В. И. Юров. – 2-е изд. – Санкт-Петербург: Питер, 2011. – 637 с.

2. Калабухов Е. В. Конструирование программ и языки программирования. Язык программирования Ассемблер : пособие / Е. В. Калабухов, И. В. Лукьянова, А. Г. Третьяков. – Минск : БГУИР, 2016. − 80 с.

3. Ружицкая, Е. А. Программирование на языке Assembler : системы счисления, программная модель микропроцессора, арифметические команды : практическое пособие / Е. А. Ружицкая, О. Г. Осипова, В. А. Ковалёва ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2016. – 47 с.

4. Ружицкая, Е. А. Программирование на языке Assembler : программирование ветвящихся и циклических вычислительных процессов, обработка массивов : практическое пособие / Е. А. Ружицкая, О. Г. Осипова, В. А. Ковалёва ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2016. – 47 с.

5. Ружицкая, Е. А. Программирование на языке Assembler : BCD-числа, цепочечные команды : практическое пособие / Е. А. Ружицкая, Е. Ю. Кузьменкова ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2017. – 47 с.

2.2 ЛАБОРАТОРНЫЙ ПРАКТИКУМ. ЧАСТЬ 2. АРХИТЕКТУРА СОПРОЦЕССОРА

ОГЛАВЛЕНИЕ

1 Архитектура сопроцессора

Сопроцессор дополняет возможности основного процессора и предназначен для выполнения следующих функций:

– поддержки арифметики с плавающей точкой;

– поддержки численных алгоритмов вычисления значений тригонометрических функций, логарифмов и т. п.;

– обработки десятичных чисел с точностью до 18 разрядов, что позволяет сопроцессору выполнять арифметические операции без округления над целыми десятичными числами со значениями до 10^{18};

– обработки вещественных чисел из диапазона $3{,}37 \cdot 10^{-4932}$ … $1.18 \cdot 10^{+4932}$.

С точки зрения программиста, сопроцессор представляет собой совокупность регистров, каждый из которых имеет свое функциональное назначение (рисунок 1.1). В программной модели сопроцессора можно выделить три группы регистров.

1. Восемь регистров r0,…,r7, составляющих основу программ- ной модели сопроцессора – *стек сопроцессора.* Размерность каждого регистра – 80 битов.

2. Три служебных регистра:

– *регистр состояния сопроцессора* swr(Status Word Register – ре-гистр слова состояния) – отражает информацию о текущем состоянии сопроцессора. В регистре swr содержатся поля, позволяющие определить: какой регистр является текущей вершиной стека сопроцессора, какие исключения возникли после выполнения последней команды, каковы особенности выполнения последней команды (некий аналог регистра флагов основного процессора) и т.д.;

– *управляющий регистр сопроцессора* cwr(Control Word Register

– регистр слова управления) – управляет режимами работы сопроцессора. С помощью полей в этом регистре можно регулировать точность выполнения численных вычислений, управлять округлением, маскировать исключения;

– *регистр слова тегов* twr (Tags Word Register – слово тегов) – используется для контроля за состоянием каждого из регистров r0,…,r7. Команды сопроцессора используют этот регистр, например, длятого чтобы определить возможность записи значений в эти регистры.

3. Два регистра указателей – данных dpr (Data Point Register) и команд ipr (Instruction Point Register). Они предназначены для запоминания информации об адресе команды, вызвавшей исключительную ситуацию, и адресе ее операнда. Эти указатели используются при обработке исключительных ситуаций (но не для всех команд).

Все эти регистры являются программно доступными.

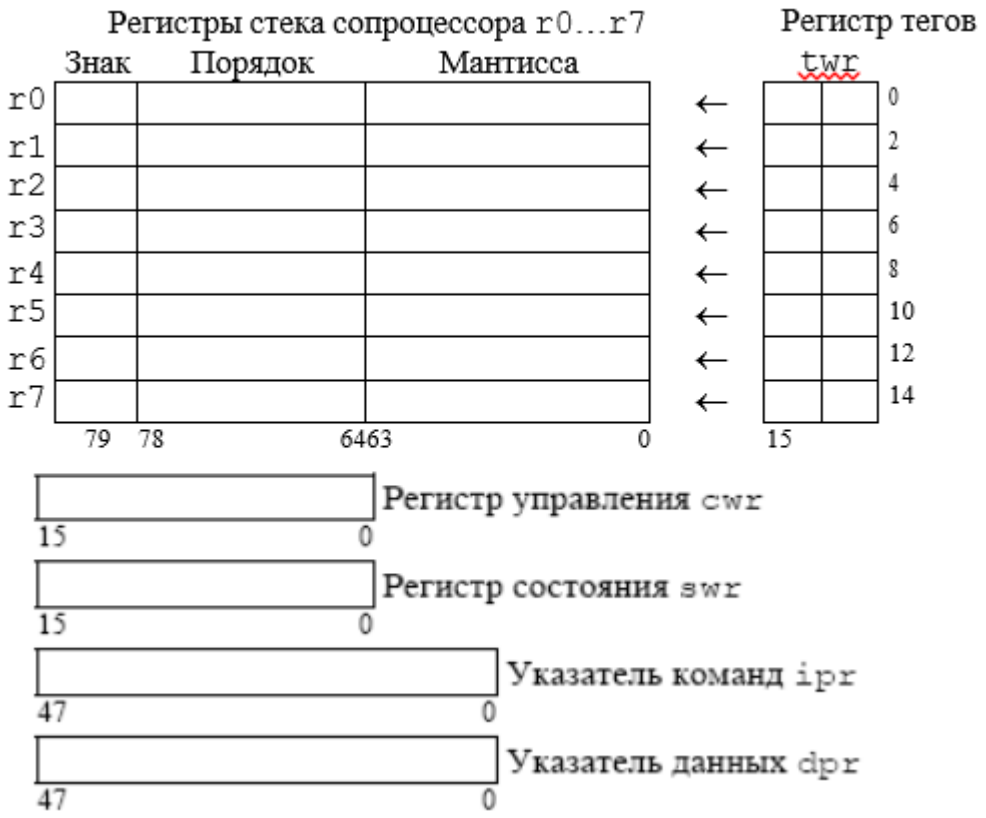


Рисунок 1.1 – Программная модель сопроцессора

Регистровый стек сопроцессора организован по принципу кольца. Это означает, что среди всех регистров, составляющих стек, нет такого, который является вершиной стека. Все регистры стека с функциональной точки зрения абсолютно одинаковы и равноправны. В стеке есть вершина, которая является плавающей. Контроль текущей вершины осуществляется аппаратно с помощью трехбитового поля top регистра swr(рисунок 1.2). В поле topфиксируется номер регистра стека 0...7 (r0,…,r7), являющегося в данный момент текущей вершиной стека.

b	c3	top			c2	c1	c0	es	sf	pe	ue	oe	ze	de	ie
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Рисунок 1.2 – Регистр состояния swr

Команды сопроцессора не оперируют физическими номерами регистров стека r0...r7. Вместо этого они используют логические номера этих регистров st(0)...st(7). С помощью логических номеров реализуется относительная адресация регистров стека сопроцессора.

Рассмотрим, каким образом «уживаются» между собой процессор и сопроцессор. Каждое из этих устройств имеет свои, несовместимые друг с другом системы команд и форматы обрабатываемых данных. Несмотря на то, что сопроцессор архитектурно представляет собой отдельное вычислительное устройство, он не может существовать отдельно от основного процессора.

Начиная с модели i486, сопроцессор и основной процессор производятся в одном корпусе и являются физически неделимыми, то есть архитектурно это по-прежнему два разных устройства, а аппаратно – одно.

Процессор и сопроцессор, являясь двумя самостоятельными вычислительными устройствами, могут работать параллельно. Но этот параллелизм касается только их внутренней работы над исполнением очередной команды. Оба процессора подключены к общей системной шине и имеют доступ к одинаковой информации. Инициирует процесс выборки очередной команды всегда основной процессор. После выборки команда попадает одновременно в оба процессора. Любая команда сопроцессора имеет код операции, первые пять бит которого имеют значение 11011. Когда код операции начинается этими битами, то основной процессор по дальнейшей содержимой коде операции выясняет, требует ли данная команда обращения к памяти. Если это так, то основной процессор формирует физический адрес операнда и обращается к памяти, после чего содержимое ячейки памяти выставляется на шину данных. Если обращение к памяти не требуется, то основной процессор заканчивает работу над данной командой (не делая попытки ее исполнения!) и приступает к декодированию следующей команды из текущего входного командного потока. Что же касается сопроцессора, то выбранная команда, как уже было отмечено, попадает в него одновременно с основным процессором. Сопроцессор, определив по первым пяти битам, что очередная команда принадлежит его системе команд, начинает ее исполнение. Если команда требовала операнд в памяти, то сопроцессор обращается к шине данных за чтением содержимого ячейки памяти, которое к этому моменту предоставлено основным процессором. Из этой схемы взаимодействия следует, что в определенных случаях необходимо согласовывать работу обоих устройств.

Например, если во входном потоке сразу за командой сопроцессора следует команда основного процессора, использующая результаты работы предыдущей команды, то сопроцессор не успеет выполнить свою команду за то время, когда основной процессор, пропустив сопроцессорную команду, выполнит свою. Очевидно, что логика работы программы будет нарушена. Возможна и другая ситуация. Если входной поток команд содержит последовательность из нескольких команд сопроцессора, то процессор, в отличие от сопроцессора, проскочит их очень быстро, чего он не должен делать, так как обеспечивает внешний интерфейс для сопроцессора.

Эти и другие более сложные ситуации приводят к необходимости синхронизации между собой работы двух процессоров. В первых моделях микропроцессоров это делалось путем вставки перед или после каждой команды сопроцессора специальной команды wait или fwait. Работа данной команды заключалась в приостановке работы основного процессора до тех пор, пока сопроцессор не заканчивал работу над последней командой. В моделях микропроцессора, начиная с i486, подобная синхронизация выполняется командами wait/fwait, которые введены в алгоритм работы большинства команд сопроцессора.

Использование сопроцессора является совершенно прозрачным для программиста. В общем случае можно воспринимать сопроцессор как набор дополнительных регистров, для работы с которыми предназначены специальные команды.

1.1 Регистр состояния swr

Регистр swr отражает текущее состояние сопроцессора после выполнения последней команды. Структурно регистр swr (рисунок 1.2) состоит из:

– 6 флагов исключительных ситуаций;

– бита sf (Stack Fault) – ошибки работы стека сопроцессора. Бит устанавливается в единицу, если возникает одна из трех исключительных ситуаций: pe, ueили ie. В частности, его установка информируето попытке записи в заполненный стек, или, напротив, попытке чтения из пустого стека. После того как этот бит проанализирован, его нужно снова установить в ноль, вместе с битами pe, ueили ie(если они были установлены);

– бита es (Error Summary) – суммарной ошибки работы сопроцессора. Бит устанавливается в единицу, если возникает любая из шести перечисленных ниже исключительных ситуаций;

– четырех битов c0–c3 (Condition Code) – кодов условия. Назначение этих битов аналогично флагам в регистре eflags основного процессора – отразить результат выполнения последней команды сопроцессора;

– трехбитного поля top. Поле содержит указатель регистра текущей вершины стека.

Почти половину регистра swr занимают биты (флаги) для регистрации исключительных ситуаций. Прерывания по месту их возникновения делятся на внешние и внутренние. Внутренние прерывания возникают в ходе работы текущей программы и делятся на синхронные (по команде int) и асинхронные, называемые *исключениями* или *особыми случаями.* Таким образом, *исключения* – это разновидность прерываний, с помощью которых процессор информирует программу о некоторых особенностях ее реального исполнения. Сопроцессор также обладает способностью возбуждения подобных прерываний при возникновении определенных ситуаций (не обязательно ошибочных). Все возможные исключения сведены к шести типам, каждому из которых соответствует один бит в регистре swr. Программисту совсем не обязательно писать обработчик для реакции на ситуацию, приведшую к некоторому исключению. Сопроцессор умеет самостоятельно реагировать на многие из них. Это так называемая обработка исключений по умолчанию. Для того чтобы «заказать» сопроцессору обработку определенного типа исключения по умолчанию, необходимо это исключение замаскировать. Такое действие выполняется с помощью установки в единицу нужного бита в управляющем регистре сопроцессора cwr (рисунок 1.3). Типы исключений, фиксируемые с помощью регистра swr:

– ie(Invalide operation Error) – недействительная операция;

– de(Denormalized operand Error) – денормализованный операнд;

– ze(divide by Zero Error) – ошибка деления на нуль;

– oe (Overflow Error) – ошибка переполнения. Возникает в случае выхода порядка числа за максимально допустимый диапазон;

– ue (Underflow Error) – ошибка антипереполнения. Возникает, когда результат слишком мал;

– pe (Precision Error) – ошибка точности. Устанавливается, когда сопроцессору приходится округлять результат из-за того, что его точное представление невозможно. Например, сопроцессору никогда не удастся точно разделить 10 на 3.

При возникновении любого из этих шести типов исключений устанавливается в единицу соответствующий бит в регистре swr, вне зависимости от того, было ли замаскировано это исключение в регистре cwrили нет.

1.2 Регистр управления cwr

Регистр управления работой сопроцессора определяет особенности обработки численных данных (рисунок 1.3). Он состоит из:

– шести масок исключений;

– поля управления точностью pc (Precision Control);

– поля управления округлением rc (Roimding Control).

Шесть масок предназначены для маскирования исключительных ситуаций, возникновение которых фиксируется с помощью шести бит регистра swr. Если какие-то биты исключений в регистре cwr установлены в единицу, то это означает, что соответствующие исключения будут обрабатываться самим сопроцессором. Если для какого-либо исключения в соответствующем бите масок исключений регистра cwr содержится нулевое значение, то при возникновении исключения этоготипа будет возбуждено прерывание 16(10h). Операционная система должна содержать (или программист должен написать) обработчик этого прерывания. Он должен выяснить причину прерывания, после чего, если это необходимо, исправить ее, а также выполнить другие действия.

				rc		pc				p	u	o	z	d	i
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Рисунок 1.3 – Регистр управления сопроцессором cwr

Поле управления точностью pcпредназначено для выбора длины мантиссы. Возможные значения в этом поле:

– pc= 00 – длина мантиссы 24 бита;

– pc= 10 – длина мантиссы 53 бита;

– pc= 11 – длина мантиссы 64 бита.

По умолчанию устанавливается значение поля pc= 11.

Поле управления округлением rc позволяет управлять округлением чисел в

процессе работы сопроцессора. Необходимость операции округления может появиться в ситуации, когда после выполнения очередной команды сопроцессора получается не представимый результат, например, периодическая дробь 3,333... Установив одно из значений в поле rc, можно выполнить округление в необходимую сторону. Для того чтобы выяснить характер округления, введем обозначения:

– m – значение в st(0)или результат работы некоторой команды, который не может быть точно представлен и поэтому должен быть округлен;

– a и b – наиболее близкие значения к значению m, которые могут быть представлены в регистре st(0) сопроцессора, причем выполняется условие $a < m < b$.

Приведем значения поля rc и опишем соответствующий им характер округления:

– 00 – значение m округляется к ближайшему числу a или b;

– 01 – значение m округляется в меньшую сторону, то есть $m = a$;

– 10 – значение m округляется в большую сторону, то есть $m = b$;

– 11 – производится отбрасывание дробной части m. Используется для приведения значения к форме, которая может использоваться в операциях целочисленной арифметики.

1.3 Регистр тегов twr

Регистр тегов twr представляет собой совокупность двухбитовых полей. Каждое двухбитовое поле соответствует определенному физическому регистру стека и характеризует его текущее состояние.

Изменение состояния любого регистра стека отражается на содержимом соответствующего этому регистру поля регистра тега. Возможны следующие значения в полях регистра тега:

– 00 – регистр стека сопроцессора занят допустимым ненулевым значением;

– 01 – регистр стека сопроцессора содержит нулевое значение;

– 10 – регистр стека сопроцессора содержит одно из специальных численных значений, за исключением нуля;

– 11 – регистр пуст и в него можно производить запись. Это значение в одном из двухбайтовых полей регистра тегов не означает, что все биты соответствующего регистра стека должны быть обязательно нулевыми.

1.4 Форматы данных

Сопроцессор расширяет номенклатуру форматов данных, с которыми работает основной процессор. Сопроцессор специально разрабатывался для вычислений с плавающей точкой. Но сопроцессор может работать и с целыми числами, хотя и менее эффективно. Форматы данных, с которыми работает сопроцессор:

– двоичные целые числа в трех форматах – 16, 32 и 64 бита;

– упакованные целые десятичные (BCD) числа – максимальная длина 18 упакованных десятичных цифр (9 байт);

– вещественные числа в трех форматах – коротком (32 бита), длинном (64 бита) и расширенном (80 бит).

Кроме этих основных форматов, сопроцессор поддерживает специальные численные значения, к которым относятся:

– денормализованные вещественные числа – это числа, меньшие минимального нормализованного числа для каждого вещественного формата, поддерживаемого сопроцессором;

– нуль;

– положительные и отрицательные значения бесконечность;

– нечисла;

– неопределенности и неподдерживаемые форматы.

В самом сопроцессоре числа в этих форматах имеют одинаковое внутреннее представление – в виде расширенного формата вещественного числа. Это один из форматов представления вещественных чисел, который точно соответствует формату регистров r0…r7 стека сопроцессора.

Таким образом, даже если используются команды сопроцессора с целочисленными операндами, то после загрузки в сопроцессор операндов целого типа они автоматически преобразуются в формат расширенного вещественного числа.

Двоичные целые числа

Сопроцессор работает с тремя типами целых чисел:

– целое слово, 16 бит, –32 768...+32 767;

– короткое целое, 32 бита $-2\cdot10^9...+2\cdot10^9$;

– длинное целое 64 бита, $-9\cdot10^{18}...+9\cdot10^{18}$.

Сопроцессор поддерживает операции с целыми числами, но работа с ними осуществляется неэффективно. Причина в том, что обработка сопроцессором целочисленных данных будет замедлена из-за необходимости выполнения дополнительного преобразования целых чисел в их внутреннее представление в виде эквивалентного вещественного числа расширенного формата.

В программе целые двоичные числа описываются с использованием директив dw, dd и dq. Например, целое число 5 может быть описано следующим образом:

ch_dw dw 5 представление в памяти: ch_dw=05 00

ch_dd dd 5 представление в памяти: ch_dw=05 00 00 00

ch_dq dq 5 представление в памяти: ch_dt=05 00 00 00 00 00 00 00

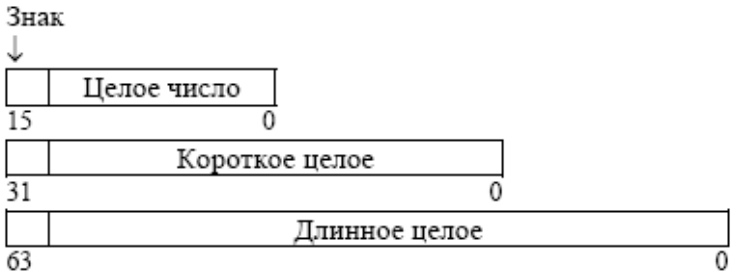


Рисунок 1.4 – Форматы целых чисел сопроцессора

Упакованные целые десятичные (BCD) числа

Сопроцессор использует один формат упакованных десятичных чисел (рисунок 1.5). Для описания упакованного десятичного числа используется директива dt. Данная директива позволяет описать 20 цифр в упакованном десятичном числе (по две в каждом байте). Из-за того, что максимальная длина упакованного десятичного числа в сопроцессоре составляет только 9 байт, в регистры r0…r7 можно поместить только 18 упакованных десятичных цифр. Старший десятый байт игнорируется. Самый старший бит этого байта используется для хранения знака числа. Например, целое число 5365904 в формате упакованного десятичного числа может быть описано следующим образом:

```
ch_dt dt 5365904
;представление в памяти: ch_dt=04 59 36 05 00 00 00 00 00 00
```

Сопроцессор имеет для работы с упакованными десятичными числами всего две команды – сохранения и загрузки.

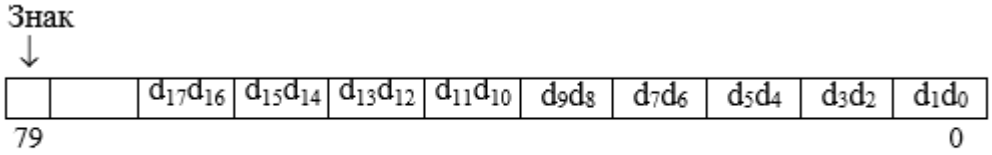


Рисунок 1.5 – Формат десятичного числа сопроцессора

Вещественные числа

Основной тип данных, с которыми работает сопроцессор, – вещественный. Данные этого типа описываются тремя форматами: коротким, длинным и расширенным (рисунок 1.6).

Рисунок 1.6 – Форматы вещественных чисел сопроцессора

Для представления вещественного числа используется формула:

$$A = \left(\pm M\right) \cdot N^{\pm(p)}, \tag{1.1}$$

где M – мантисса числа A. Мантисса должна удовлетворять условию $|M| < 1$;

N – основание системы счисления, представленное целым положительным числом;

p – порядок числа, показывающий истинное положение точки в разрядах мантиссы (по этой причине вещественные числа имеют еще название чисел с плавающей точкой, так как ее положение в разрядах мантиссы зависит от значения порядка).

Для удобства обработки в компьютере чисел с плавающей точкой, архитектурой компьютера на компоненты формулы (1.1) накладываются следующие ограничения:

– основание системы счисления $N = 2$;

– мантисса M должна быть представлена в *нормализованном* виде. Для архитектуры микропроцессора Intel нормализованным является число вида:

$$A = (-1)^s \cdot N^q \cdot M, \tag{1.2}$$

где s – значение знакового разряда (0 – число больше нуля; 1 – число меньше нуля);

p – порядок числа. Его значение аналогично значению порядка p в формуле (1.1).

В этой формуле знак имеют и порядок вещественного числа, и его мантисса. На рисунке 1.6 видно, что формат хранения вещественного числа в памяти имеет только поле для знака мантиссы. А где же хранится знак порядка? В сопроцессоре Intel на аппаратном уровне принято соглашение, что порядок p определяется в формате вещественного числа особым значением, называемым *характеристикой* q. Величина q связана с порядком p посредством формулы (1.3) и представляет собой некоторую константу. Условно назовем ее *фиксированным смещением*:

$$q = p + \textit{фиксированное смещение}. \tag{1.3}$$

Для каждого из трех возможных форматов вещественных чисел смещение q имеет разное, но фиксированное для конкретного формата значение, которое зависит от количества разрядов, отводимых под характеристику (таблица 1.1).

В таблице 1.1 показаны диапазоны значений характеристик q и соответствующих им истинных порядков p вещественных чисел. Нулевому порядку вещественного числа в коротком формате соответствует значение характеристики равное 127, которому в двоичном представлении соответствует значение 0111 1111. Отрицательному порядку p, например, –1, будет соответствовать характеристика $q = –1 + 127 = 126$.

Таблица 1.1 – Формат вещественных чисел

Формат	Короткий	Длинный	Расширенный
Длина числа (биты)	32	64	80
Размерность мантиссы M	24	53	64
Диапазон значений	$10^{-38}...10^{38}$	10^{-308} 10^{308}	10^{-4932} 10^{4932}
Размерность характеристики q	8	11	15
Значение фиксированного смещения	+127 (7F) 0111 1111	+1023 (3FF) 0011 1111 1111	+16383 (3FFF) 0011 1111 1111 1111
Диапазон характеристик q	0...255	0...2047	0...32767
Диапазон порядков p	–126...+127	–1022...+1023	–16382...+16383

В двоичном виде ей соответствует значение 0111 1110. Положительному порядку p, например, +1, будет соответствовать характеристика $q = 1 + 127 = 128$, в двоичном виде ей соответствует значение 1000 0000. То есть, все положительные порядки имеют в двоичном представлении характеристики старший бит равный единице, а отрицательные порядки – нет. Знак порядка спрятан в старшем бите характеристики.

Так как нормализованное вещественное число всегда имеет целую единичную часть (исключая представление перечисленных выше специальных численных значений), то при его представлении в памяти появляется возможность считать первый разряд вещественного числа единичным по умолчанию и учитывать его наличие только па аппаратном уровне. Это дает возможность увеличить диапазон представимых чисел, так как появляется лишний разряд, который можно использовать для представления мантиссы числа. Но это справедливо только для короткого и длинного форматов вещественных чисел. Расширенный формат, как внутренний формат представления числа любого типа в сопроцессоре, содержит целую единичную часть вещественного в явном виде.

Короткое вещественное число длиной в 32 разряда определяется директивой dd. При этом обязательным в записи числа является наличие десятичной точки, даже если оно не имеет дробной части. Для транслятора десятичная точка является указанием, что число нужно представить в виде числа с плавающей точкой в коротком формате. Это же касается длинного и расширенного форматов представления вещественных чисел, определяемых директивами dqи dt.

Другой способ задания вещественного числа директивами dd, dq и dt– экспоненциальная форма с использованием символа «e».

Пример. Определим в программе вещественное число 45,56 в коротком формате:

```
dd 45.56
или
dd 45.56e0
или
dd 0.4556e2
```

В памяти это число будет выглядеть так:

```
71 3d 36 42
```

Так как в архитектуре Intel принят перевернутый порядок следования байт в памяти в соответствии с принципом «младший байт по младшему адресу», истинное представление числа 45,56 будет следующим: 42 36 3d 71. В двоичном представлении в памяти число будет иметь вид, представленный на рисунке 1.7. На рисунке 1.7 видно, что старшая единица мантиссы при представлении в памяти отсутствует.

Переведем десятичную дробь 45,56 в эквивалентное двоичное представление:

$45{,}56_{10} = 101101{,}10001111010111000 1\ldots_2$.

Нормализуем число. Для этого переносим точку влево, до тех пор, пока в целой части числа не останется одна двоичная единица. Число переносов влево (или вправо, если десятичное число было меньше единицы) будет являться порядком числа. После перемещения точки получаем значение порядка $p = 5$. Соответственно, характеристика будет выглядеть так: $q = p + 127 = 5 + 127 = 132_{10} = 1000\ 0100_2$.

Сформированный результат в виде вещественного числа в коротком формате состоит из трех компонент:

– знака – 0;

– характеристики 1000 0100;

– мантиссы 0110 1100 0111 1010 1110 001…, содержащей целую чисть числа 45 без первой значащей единицы (01101) и дробную часть числа (100 0111 1010 1110 001…).

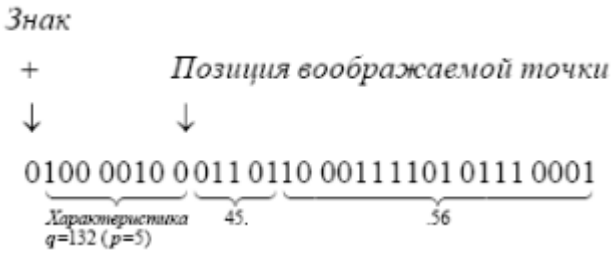


Рисунок 1.7 – Двоичное представление вещественного числа в коротком формате

Пример. Определим в программе вещественное число 45,56 в длинном формате (рисунок 1.8):

dq 45.56

или

dq 45.56e0

В памяти это число будет выглядеть так:

47 e1 7a 14 ae c7 46 40

Перевернув его, получим истинное значение:

40 46 c7 ae 14 7a el 47

Для представления числа 45,56 в регистрах сопроцессора переведем это число в двоичную систему счисления:

$45{,}56_{10} = 101101{,}10001111010111000 1\ldots_2$.

Порядок числа $p = 5$.

Характеристика числа

$q = p + 1023 = 5 + 1023 = 1028_{10} = 100\ 0000\ 0100_2$.

Сформированный результат в виде вещественного числа в длинном формате состоит из трех компонент:

– знака – 0;

– характеристики 100 0000 0100;

– мантиссы 0110 1100 0111 1010 1110 001…, содержащей целую чисть числа 45 без первой значащей единицы (01101) и дробную часть числа (100 0111 1010 1110 001…).

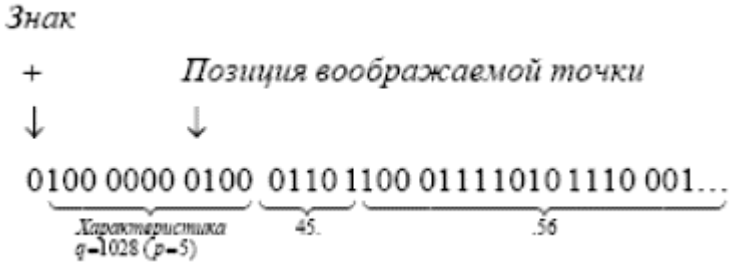


Рисунок 1.8 – Двоичное представление вещественного числа в длинном формате

Пример. Определим в программе вещественное число 45,56 в расширенном

формате (рисунок 1.9):

dt 45.56

В памяти это число будет выглядеть так:

71 3d 0a d7 a3 70 3d b6 04 40

Перевернув его, получим истинное значение в памяти:

40 04 b6 3d 70 a3 d7 0a 3d 71

Для представления числа 45,56 в регистрах сопроцессора переведем это число в двоичную систему счисления:

$45{,}56_{10} = 101101{,}10001111010111000 01\ldots_2$.

Характеристика числа

$q = p + 16383 = 5 + 16383 = 16388_{10} = 100\ 0000\ 000\ 0100_2$.

Сформированный результат в виде вещественного числа в длинном формате состоит из трех компонент:

– знака – 0;

– характеристики 100 0000 0000 0100;

– мантиссы 1011 0110 0011 1101 0111 0001…, содержащей целую часть числа 45 с первой значащей единицей (101101) и дробную часть числа (100 0111 1010 1110 001…).

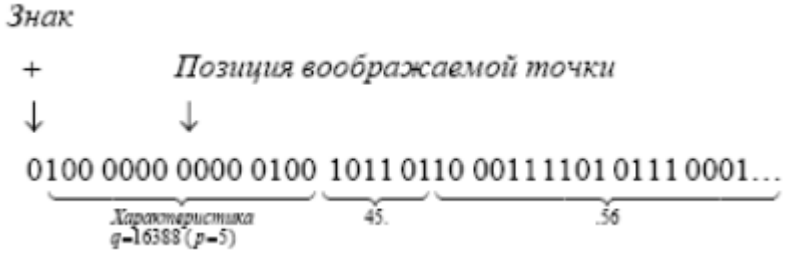


Рисунок 1.9 – Двоичное представление вещественного числа в расширенном формате

В мантиссе явно присутствует старшая единица, чего не было в коротком и длинном форматах представления вещественного числа.

Специальные численные значения

Несмотря на большой диапазон вещественных значений, представимых в регистрах стека сопроцессора, понятно, что бесконечное количество их значений находится за рамками этого диапазона. Для того чтобы иметь возможность реагировать на вычислительные ситуации, в которых возникают такие значения, в сопроцессоре и предусмотрены специальные комбинации бит, называемые *специальными численными значениями.* При необходимости программист может сам кодировать специальные численные значения. Это возможно потому, что вещественные числа, описанные директивой dt, соответствующие команды сопроцессора загружают без всяких преобразований.

Денормализованные вещественные числа – это числа, которые меньше минимального нормализованного числа для каждого вещественного формата.

Например, для вещественного числа в расширенном формате диапазон представимых значений в сопроцессоре показан на рисунке 1.10.

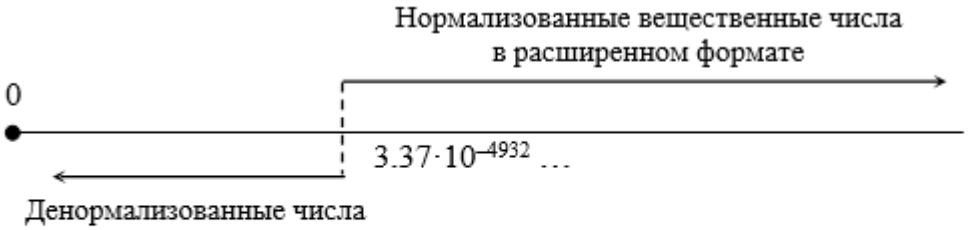


Рисунок 1.10 – Положение денормализованных вещественных чисел на числовой шкале

Как известно, сопроцессор хранит числа в нормализованном виде. По мере приближения чисел к нулю ему все труднее «вытягивать» их значения к нормализованному виду, то есть к такому виду, чтобы первой значащей цифрой мантиссы была единица. Размерность разрядной сетки, отведенной в форматах вещественных чисел сопроцессора, для представления характеристики не безгранична. Поэтому при определенных значениях числа в расширенном формате значение характеристики становится равным нулю (рисунок 1.10). Но на самом деле число отлично от нуля, так как это все же не настоящий численный нуль. Таким образом, между истинным нулем и минимально представимым нормализованным числом есть еще бесконечное количество очень маленьких чисел. Это и есть так называемые денормализованные числа. Они имеют нулевой порядок и ненулевую мантиссу. Диапазон представимых в сопроцессоре денормализованных чисел не безграничен, так как количество разрядов мантиссы ограниченно (рисунок 1.11).

При формировании денормализованного значения в некотором регистре стека в соответствующем этому регистру теге регистра twr формируется специальное значение (10).

Рисунок 1.11 – Диапазон представимых в сопроцессоре денормализованных чисел

Нуль. Значение *нуля* также относят к специальным численным значениям. Это делается из-за того, что это значение особо выделяется среди корректных

вещественных значений, формируемых как результат работы некоторой команды. Более того, нуль может формироваться как реакция сопроцессора на определенную вычислительную ситуацию.

Значение истинного нуля может иметь знак (рисунок 1.12). Если необходимо определить знак нуля, то используется команда fxam. В результате работы этой команды в бит c1 регистра swr заносится знак операнда. При загрузке нуля в регистр стека в соответствующем теге регистра twr формируется специальное значение (01).

Значение нуля может быть сформировано в результате возникновения ситуации антипереполнения, а также при работе команд с нулевыми операндами.

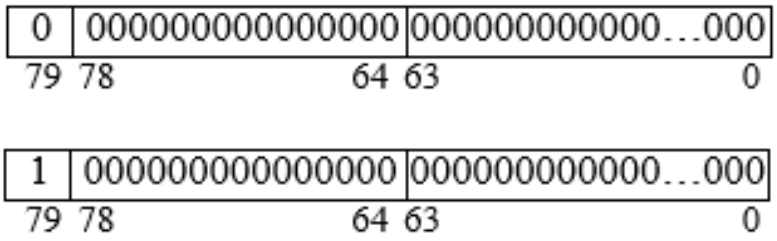


Рисунок 1.12 – Представление нуля в регистре стека сопроцессора

Значение бесконечность. Сопроцессор имеет средства для представления значения бесконечность. Это значение формируется с помощью специальных битовых значений. Формат регистра стека сопроцессора, содержащего значение бесконечность, приведен на рисунке 1.13. Значение бесконечность может иметь знак, при этом значения мантиссы и характеристики фиксированы. Именно в этом заключается отличие значения бесконечность от остальных специальных значений. Среди причин, приводящих к формированию значения бесконечность, можно выделить переполнение и деление на нуль. При формировании значения бесконечность в некотором регистре стека, в соответствующем теге регистра twr формируется специальное значение (10).

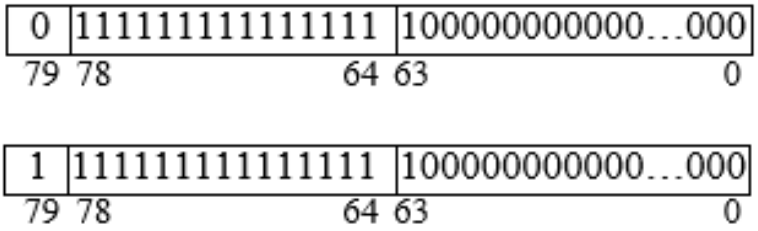


Рисунок 1.13 – Представление значения бесконечность в регистре стека сопроцессора

Нечисла. К *нечислам* относятся такие битовые последовательности в

регистре стека сопроцессора, которые не совпадают ни с одним из рассмотренных выше форматов значений. Нечисло должно иметь единичную мантиссу и любую мантиссу, кроме 100...00, которая зарезервирована для значения бесконечность. Различают два типа нечисел:

– SNAN(Signaling Noil a Number) – сигнальные нечисла;

– QNAN(Quiet Non A Number) – спокойные (тихие) нечисла.

Сигнальное нечисло – битовое значение с единичным значением поля характеристики и мантиссы, первый бит которой, следующий за первым единичным значащим битом мантиссы, равен нулю (рисунок 1.14, *а*). Сопроцессор реагирует на появление этого числа в регистре стека возбуждением исключения недействительная операция. Программисты могут формировать эти числа в регистре стека сопроцессора преднамеренно, например, для того чтобы искусственно возбудить в нужной ситуации указанное исключение. Именно по этой причине данные числа называются сигнальными. Если снять маску у флага недействительная операция в регистре cwr, то будет вызван обработчик, который выполнит заданные программистом действия.

Спокойное нечисло – битовое значение с единичным значением полей характеристики и мантиссой, первые два бита которой равны единице (рисунок 1.14, *б*).

При формировании нечисла в некотором регистре стека в сооветствующем теге регистра twrформируется специальное значение (10).

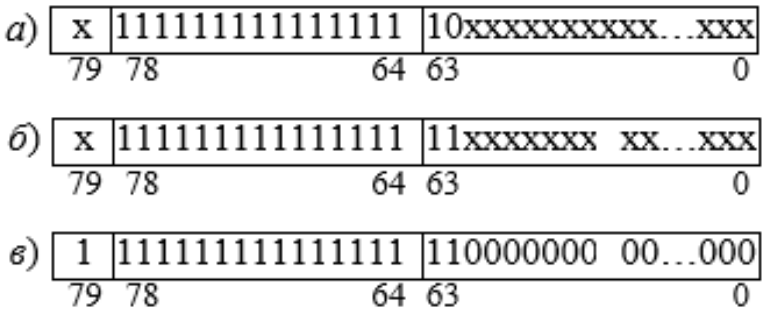


Рисунок 1.14 – Представление нечисел в регистре стека сопроцессора

Сопроцессор самостоятельно не формирует сигнальных чисел, но в качестве реакции па определенные исключения он может формировать спокойные нечисла, например, нечисло *вещественная неопределенность*. Его значение показано на рисунке 1.14, *в*. Вещественная неопределенность формируется как маскированная реакция сопроцессора на исключение недействительная операция. Другие спокойные нечисла могут формироваться после выполнения команд, в которых хотя бы один из операндов был спокойным нечислом. Это может породить цепную

реакцию, которая приведет к ошибочному результату. Поэтому в процессе вычислений рекомендуется периодически контролировать результаты исполнения команд на предмет появления спокойных нечисел.

2 Программная модель сопроцессора. Регистры

2.1 Система команд сопроцессора

Система команд сопроцессора включает около 80 машинных команд.

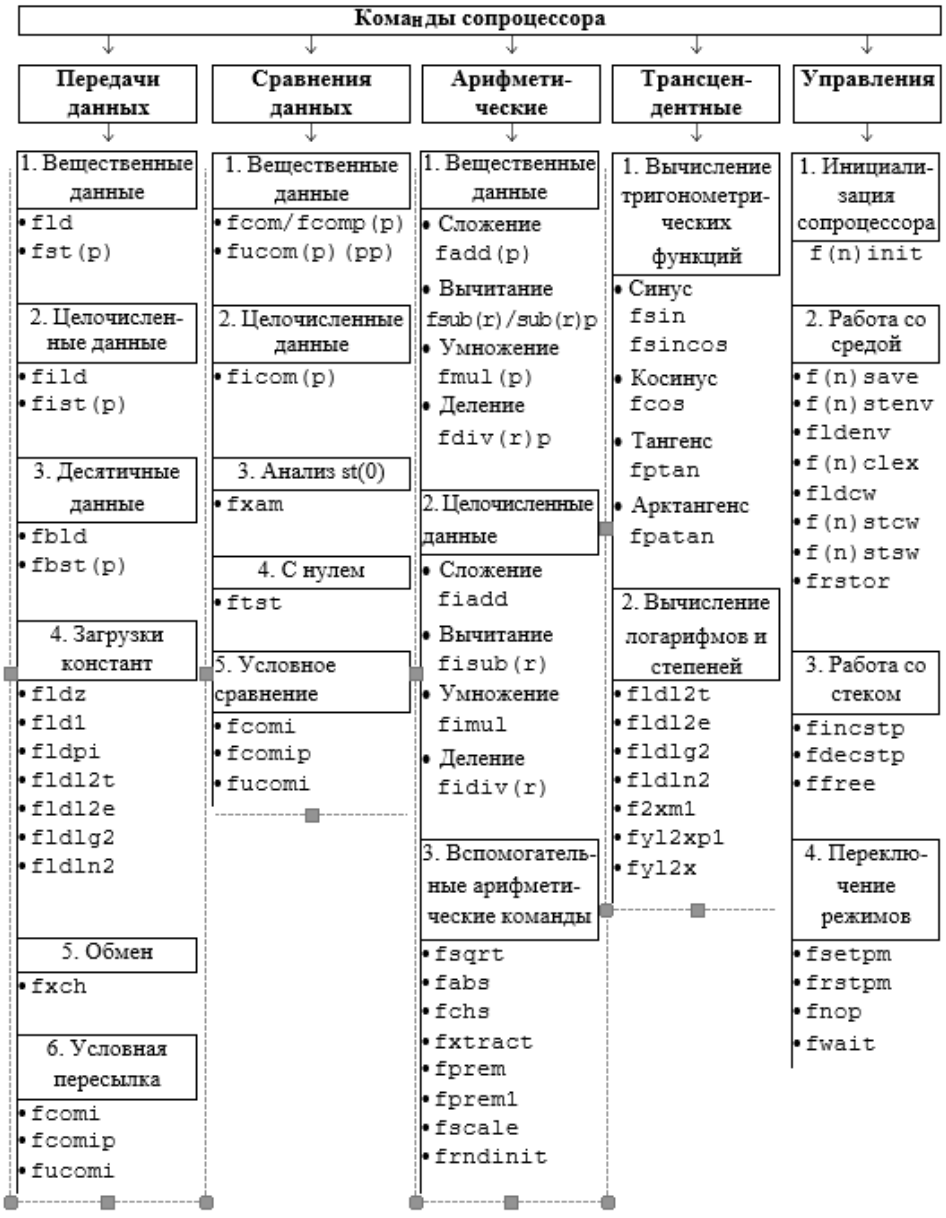


Рисунок 2.1 – Функциональная классификация команд сопроцессора

Мнемоническое обозначение команд сопроцессора характеризует особенности их работы:

1. Все мнемонические обозначения начинаются с символа f(float).
2. Вторая буква мнемонического обозначения определяет тип операнда в

памяти, с которым работает команда:

i– целое двоичное число;

b– целое десятичное число;

отсутствие буквы – вещественное число.

3. Последняя буква p в мнемоническом обозначении команды означает, что последним действием команды обязательно является извлечение операнда из стека.

4. Последняя или предпоследняя буква r (reversed) в мнемоническом обозначении команды означает реверсивное следование операндов при выполнении команд вычитания и деления, так как для них важен порядок следования операндов.

Система команд сопроцессора отличается большой гибкостью в выборе вариантов задания команд, реализующих определенную операцию, и их операндов.

Минимальная длина команды сопроцессора – 2 байта.

Методика написания программ для сопроцессора имеет свои особенности. Главная причина – в стековой организации сопроцессора.

При разработке программ необходимо учитывать:

– ограниченность глубины стека сопроцессора;

– несовпадение форматов операндов;

– отсутствие поддержки на уровне команд сопроцессора некоторых операций, таких как возведение в степень, вычисление тригонометрических функций.

2.2 Команды передачи данных

Группа команд передачи данных предназначена для организации обмена между регистрами стека, вершиной стека сопроцессора и ячейками оперативной памяти.

Команды этой группы имеют такое же значение для программирования сопроцессора, как команда mov – для программирования основного процессора. Главной функцией всех команд загрузки данных в сопроцессор является преобразование данных к единому представлению в виде вещественного числа расширенного формата. Это же касается и обратной операции – сохранения в памяти данных из сопроцессора.

Команды передачи данных можно разделить на 3 группы:

1) команды передачи данных в вещественном формате;
2) команды передачи данных в целочисленном формате;
3) команды передачи данных в десятичном формате.

Команды передачи данных в вещественном формате

fld источник – загрузка вещественного числа из области памяти на вершину стека сопроцессора.

fst приемник – сохранение вещественного числа из вершины стека

сопроцессора в память. Как следует из анализа мнемокода команды (отсутствует символ p), сохранение числа в памяти не сопровождается выталкиванием его из стека, то есть текущая вершина стека сопроцессора не меняется (поле top не меняется).

fstp приемник – сохранение вещественного числа из вершины стека сопроцессора в память. В отличие от предыдущей команды, в конце мнемонического обозначения данной команды присутствует символ p, что означает выталкивание вещественного числа из стека после его сохранения в памяти. Команда изменяет поле top, увеличивая его на единицу. Вследствие этого вершиной стека становится следующий больший по своему физическому номеру регистр стека сопроцессора.

Команды передачи данных в целочисленном формате

fild источник – загрузка целого числа из памяти на вершину стека сопроцессора.

fist приемник – сохранение целого числа из вершины стека сопроцессора в память. Сохранение целого числа в памяти не сопровождается выталкиванием его из стека, то есть текущая вершина стека сопроцессора не изменяется.

fistp приемник – сохранение целого числа из вершины стека в память. Аналогично сказанному ранее о команде fstp, последним действием команды является выталкивание числа из стека с одновременным преобразованием его в целое значение.

Команды передачи данных в десятичном формате

fbld источник – загрузка десятичного числа из памяти в вершину стека сопроцессора.

fbstp приемник – сохранение десятичного числа из вершины стека сопроцессора в области памяти. Значение выталкивается из стека после преобразования его в формат десятичного числа.

Для десятичных чисел нет команды сохранения значения в памяти без выталкивания из стека.

Команда обмена вершины регистрового стека st(0) с любым другим регистром стека сопроцессора st(i)

fxch st(i)

Действие команд загрузки fld, fild и fbld можно сравнить с командой push основного процессора. Аналогично ей (push уменьшает значение в регистре sp) команды загрузки сопроцессора перед сохранением значения в регистровом стеке сопроцессора вычитают из содержимого поля topрегистра состояния swединицу. Это означает, что вершиной стека становится регистр с физическим номером на единицу меньше.

***Для наблюдения за состоянием регистров сопроцессора используется окно Numeric processor* (*View – Numeric Processor*).**

Команды загрузки констант

Основным назначением сопроцессора является поддержка вычислений с плавающей точкой. В математических вычислениях довольно часто встречаются предопределенные константы, и сопроцессор хранит значения некоторых из них.

Для каждой предопределенной константы существует специальная команда, которая производит загрузку ее на вершину регистрового стека сопроцессора:

fldz – загрузка нуля; fld1– загрузка единицы; fldpi– загрузка числа □;

fldl2t– загрузка двоичного логарифма десяти;

fldl2e– загрузка двоичного логарифма экспоненты (числа *e*);

fldlg2– загрузка десятичного логарифма двойки;

fldln2– загрузка натурального логарифма двойки.

2.3 Команды сравнения данных

Команды сравнения данных сравнивают значение числа в вершине стека и операнда, указанного в команде.

fcom [операнд_в_памяти]– команда без операндов сравнивает два значения: одно находится в регистре st(0), другое в регистре st(1). Если указан операнд [операнд_в_памяти], то сравнивается значение в регистре st(0) стека сопроцессора со значением в памяти. fcomp операнд – команда сравнивает значение в вершине стека сопроцессора st(0) со значением операнда, который находится в регистре или в памяти. Последним действием команды является выталкивание значения из st(0).

fcompp операнд – команда аналогична по действию команде fcom без операндов, но последним ее действием является выталкивание из стека значений обоих регистров, st(0) и st(1).

ficom операнд_в_памяти – команда сравнивает значение в вершине стека сопроцессора st(0) с целым операндом в памяти. Длина целого операнда – 16 или 32 бита.

ficomp операнд – команда сравнивает значение в вершине стека сопроцессора st(0) с целым операндом в памяти. После сравнения и установки битов c3...c0 команда выталкивает значение из st(0). Длина целого операнда – 16 или 32 бита.

ftst – команда не имеет операндов и сравнивает значения в st(0) с нулем (значением 00).

Предыдущие команды сравнения работают корректно, если операнды в них являются целыми или вещественными числами. Когда один из операндов оказывается нечислом, то фиксируется исключение недействительной ситуации, а коды условия c3...c0 соответствуют исключительной ситуации несравнимых или неупорядоченных операндов. Само же действие сравнения не производится.

Процессор предоставляет три команды, позволяющие все же произвести

сравнение таких операндов, но как вещественных чисел без учета их порядков.

fucom st(i) – команда сравнивает значения (без учета их порядков) в регистрах стека сопроцессора st(0)и st(i).

fucomp st(i) – команда сравнивает значения (без учета их порядков) в регистрах стека сопроцессора st(0) и st(i). Последним действием команды является выталкивание значения из вершины стека.

fucompp st(i) – команда сравнивает значения (без учета их порядков) в регистрах стека сопроцессора st(0) и st(i). Последние два действия команды – выталкивание значения из вершины стека.

В результате работы команд сравнения в регистре состояния уста навливаются следующие значения битов кода условия c3, c2, c0:

- если st(0)> операнда, то 000;
- если st(0)< операнда, то 001;
- если st(0)= операнду, то 100;
- если операнды неупорядочены, то 111.

Для того чтобы получить возможность реагировать на эти коды командами условного перехода основного процессора (они реагируют на флаги в eflags), нужно записать сформированные биты условия c3, c2, c0 в регистр eflags. В системе команд сопроцессора существует команда fstsw, которая позволяет запомнить слово состояния сопроцессора в регистре ax или ячейке памяти. Далее значения нужных битов извлекаются и анализируются командами основного процессора.

Например, запись старшего байта слова состояния, в котором находятся биты c0...c3, в младший байт регистра eflags/flags осуществляется командой sahf. Эта команда записывает содержимое ahв младший байт регистра eflags/flags. После этого бит c0 записывается на место флага cf, c2– на место pf, c3– на место zf. Бит c1 выпадает из общего правила, так как в регистре флагов на месте соответствующего ему бита находится единица. Анализ этого бита нужно проводить с помощью логических команд основного процессора. Таким образом, нужно применять те команды условного перехода, которые анализируют состояние указанных флагов.

К группе команд сравнения данных логично отнести и команду fxam, которая анализирует операнд в вершине стека сопроцессора st(0) и формирует значение битов c0, cl, c2, c3 в регистре состоя ния сопроцессора swr. По состоянию этих битов можно судить:

- о знаке мантиссы – знаковый бит операнда в st(0) заносится в бит c0регистра swr;
- корректности записи вещественного числа в st(0) – идентифицируются пустой регистр, корректное вещественное число, нечисло и неизвестный формат;
- типе специального численного значения: бесконечность, нуль, денормализованный операнд.

Пример. Программа разбиения массива вещественных чисел в формате двойного слова на два массива. В первый массив помещаются все элементы, которые больше или равны нулю, а во второй – меньше нуля.

```
;Исследование команд сравнения данных
.586p
masm
model use16 small
.stack 100h
;сегмент данных
.data
;исходный массив
mas dd -2.0, 45.7, -9.4, 7.3, 60.3, -58.44, 890e7, -98746e3
mas_h_0 dd 8 dup (0)
;массив значений больше либо равных 0
i_mas_h_0 dd 0 ;текущий индекс в mas_h_0
mas_l_0 dd 8 dup (0) ;массив значений меньших 0
i_mas_l_0 dd 0 ;текущий индекс в mas_l_0
.code
main proc ;начало процедуры main
mov ax, @data
mov ds, ax
xor esi, esi
mov cx, 8  ;счетчик циклов
finit
  ;приведение сопроцессора в начальное состояние
fldz ;загрузка нуля в st(0)
cycl:
fcom mas[esi*4]
  ;сравнение нуля в st(0) с очередным
  ;элементом массива mas
fstsw ax ;сохранение swr в регистре ax
sahf ;запись swr->ax-> регистр флагов
jp error ;переход по "плохому" операнду
  ;в команде fcom
jc hi_0 ;переход, если mas[esi*4]>= 0
  ;(mas[esi*4]>=st(0))
  ;пересылка операнда mas[esi*4]
  ;меньшего 0 в массив mas_l_0
mov eax, mas[esi*4]
mov edi, i_mas_l_0
mov mas_l_0[edi*4], eax
```

```
inc i_mas_l_0
jmp cycl_bst
hi_0:
  ;пересылка операнда mas[esi*4] большего
  ;или равного 0 в массив mas_h_0mov eax, mas[esi*4]
mov edi, i_mas_h_0
mov mas_h_0[edi*4], eaxinc i_mas_h_0
cycl_bst:
inc si
loop cycl
error:
  ;здесь можно вывести сообщение об ошибке
  ;в задании операндов
exit:
mov ax, 4c00hint 21h
main endp
end main
```

2.4 Арифметические команды

Команды сопроцессора, входящие в группу арифметических команд, реализуют четыре основные арифметические операции – сложение, вычитание, умножение и деление.

С точки зрения типов операндов арифметические команды сопроцессора можно разделить на команды, работающие с вещественными и целыми числами.

Целочисленные арифметические команды

Целочисленные арифметические команды предназначены для работы на тех участках вычислительных алгоритмов, где в качестве исходных данных используются целые числа в памяти в форматах слово и короткое слово, имеющие размерность 16 и 32 бита.

fiadd источник – команда складывает значения st(0) и целочисленного источника, в качестве которого выступает 16- или 32-разрядный операнд в памяти. Результат сложения запоминается в регистре стека сопроцессора st(0) (st(0)= st(0)+И).

fisub источник – команда вычитает значение целочисленного источника из st(0). Результат вычитания запоминается в регистре стека сопроцессора st(0). В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти.

fimul источник– команда умножает значение целочисленного источника на содержимое st(0). Результат умножения запоминается в регистре стека сопроцессора st(0). В качестве источника выступает 16- или 32-разрядный

целочисленный операнд в памяти.

fidiv источник – команда делит содержимое st(0) на значение целочисленного источника. Результат деления запоминается в регистре стека сопроцессора st(0). В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти.

Для команд, реализующих арифметические действия деления и вычитания, важен порядок расположения операндов. По этой причине система команд сопроцессора содержит соответствующие реверсивные команды, повышающие удобство программирования вычислительных алгоритмов. Чтобы отличить эти команды от обычных команд деления и вычитания, их мнемокоды оканчиваются символом r.

fisubr источник – команда вычитает значение st(0) из целочисленного источника. Результат вычитания запоминается в регистре стека сопроцессора st(0). В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти.

fidivr источник – команда делит значение целочисленного источника на содержимое st(0). Результат деления запоминается в регистре стека сопроцессора st(0). В качестве источника выступает 16- или 32-разрядный целочисленный операнд в памяти.

Пример программы вычисления значения выражения:

$$u = \begin{cases} \dfrac{x+y}{a}, & a \neq 0; \\ x+y, & a = 0. \end{cases}$$

Все переменные *x*, *y* и *a* целого типа в формате слова. Результат необходимо сохранить в ячейке памяти в формате десятичного числа.

```
;Исследование целочисленных арифметических команд
.586p
masm
model use16 small
.stack 100h
.data
  ;сегмент данных
  ;исходные данные
a dw 0
x dw 8
y dw 4
u dt 0
  ;сегмент кода
.code
main proc
mov ax, @data
mov ds, ax finit
```

```
  ;приведение сопроцессора в начальное состояние
fild a ;загрузка значение a в st(0)
fxam ;определяем тип a
fstsw ax ;сохранение swr в регистре ax
sahf ;запись swr->ax-> регистр флагов
jc no_null
jp no_null
jnz no_null
  ;вычисление формулы u=x+y:
fild x
fiadd y
fbstp u
jmp exit
no_null:
  ;вычисление формулы u=(x-y)/a:
  fild x
fisub y
fidiv a
fbstp u
exit:
mov ax, 4c00h
int 21h
main endp
end main
```

Вещественные арифметические команды

Схема расположения операндов вещественных команд традиционна для команд сопроцессора. Один из операндов располагается в вершине стека сопроцессора – регистре st(0), куда после выполнения команды записывается и результат, а второй операнд может быть расположен либо в памяти, либо в другом регистре стека сопроцессора. Допустимыми типами операндов в памяти являются все перечисленные ранее вещественные форматы за исключением расширенного.

В отличие от целочисленных арифметических команд, вещественные арифметические команды допускают большее разнообразие в сочетании местоположения операндов и самих команд для выполнения конкретного арифметического действия.

Команды сложения

fadd – команда складывает значения в st(0) и st(1). Результат сложения запоминается в регистре стека сопроцессора st(0).

fadd источник – команда складывает значения st(0) и источника, представляющего адрес ячейки памяти. Результат сложения запоминается в

регистре стека сопроцессора st(0).

fadd st(i),st – команда складывает значение в регистре стека сопроцессора st(i) со значением в вершине стека st(0). Результат сложения запоминается в регистре st(i).

faddp st(i),st – команда производит сложение вещественных операндов аналогично команде fadd st(i),st, однако послед- ним действием команды является выталкивание значения из вершины стека сопроцессора st(0). Результат сложения остается в регистре st(i–1).

Команды вычитания

fsub – команда вычитает значение в st(1) из значения в st(0). Результат вычитания запоминается в регистре стека сопроцессора st(0).

fsub источник – команда вычитает значение источника из значения в st(0). Источник представляет адрес ячейки памяти, содержащей допустимое вещественное число. Результат сложения запоминается в регистре стека сопроцессора st(0).

fsub st(i),st – команда вычитает значение в вершине стека st(0) из значения в регистре стека сопроцессора st(i). Результат вычитания запоминается в регистре стека сопроцессора st(i).

fsubp st(i),st – команда вычитает вещественные операнды аналогично команде fsub st(i),st. Последним действием коман- ды является выталкивание значения из вершины стека сопроцессора st(0). Результат вычитания остается в регистре st(i-1).

fsubr st(i),st – команда вычитает значение в вершине стека st(0) из значения в регистре стека сопроцессора st(i). Результат вычитания запоминается в вершине стека сопроцессора – регистре st(0).

fsubrp st(i),st – команда производит вычитание подобно команде fsubr st(i),st. Последним действием команды является выталкивание значения из вершины стека сопроцессора st(0). Результат вычитания остается в регистре st(i-1).

Команды умножения вещественных операндов. Операнды располагаются исключительно в стеке сопроцессора.

fmul– команда не имеет операндов. Умножает значения в st(0)на содержимое в st(1). Результат умножения запоминается в регистре стека сопроцессора st(0).

fmul st(i) – команда умножает значение в st(0) на содержимое регистра стека st(i). Результат умножения запоминается в регистре стека сопроцессора st(0).

fmul st(i),st – команда умножает значения в st(0) на содержимое произвольного регистра стека st(i). Результат умножения запоминается в регистре стека сопроцессора st(i).

fmulp st(i),st – команда производит умножение подобно команде fmul st(i),st. Последним действием команды является выталкивание значения из вершины стека сопроцессора st(0). Результат умножения остается в регистре st(i-1).

Команды деления вещественных данных. Операнды этих команд располагаются в стеке сопроцессора:

fdiv – команда (без операндов) делит содержимого регистра st(0) на значение регистра сопроцессора st(1). Результат деления запоминается в регистре стека сопроцессора st(0).

fdiv st(i) – команда делит содержимое регистра st(0) на содержимое регистра сопроцессора st(i). Результат деления запоминается в регистре стека сопроцессора st(0).

fdiv st(i),st – команда производит деление аналогично ко- манде fdiv st(i), но результат деления запоминается в регистре стека сопроцессора st(i).

fdivp st(i),st – команда производит деление аналогично команде fdiv st(i),st. Последним действием команды является выталкивание значения из вершины стека сопроцессора st(0). Результат деления остается в регистре st(i-1).

fdivr st(i),st – команда делит содержимое регистра st(i) на содержимое вершины регистра сопроцессора st(0). Результат деления запоминается в регистре стека сопроцессора st(0).

fdivrp st(i),st – команда делит содержимое регистра st(i) на содержимое вершины регистра сопроцессора st(0). Результат деления запоминается в регистре стека сопроцессора st(i), после чего производится выталкивание содержимого st(0) из стека. Результат деления остается в регистре st(i-l).

Дополнительные арифметические команды

Далее перечислены команды, входящие в группу дополнительных арифметических команд:

fsqrt – вычисление квадратного корня из значения, находящегося в вершине стека сопроцессора – регистре st(0). Команда не имеет операндов. Результат вычисления помещается в регистр st(0).

fabs – вычисление модуля значения, находящегося в вершине стека сопроцессора – регистре st(0). Команда не имеет операндов. Результат вычисления помещается в регистр st(0).

fchs – изменение знака значения, находящегося в вершине стека сопроцессора – регистре st(0). Команда не имеет операндов. Результат вычисления помещается обратно в регистр st(0). Отличие команды fchs от команды fabs в том, что команда fchs только инвертирует знаковый разряд значения в регистре st(0), не меняя значения остальных битов. Команда вычисления модуля fabs при наличии отрицательного значения в регистре st(0), наряду с инвертированием знакового разряда, выполняет изменение остальных битов значения таким образом, чтобы в st(0) получилось соответствующее положительное число.

fxtract – команда выделения порядка и мантиссы значения, находящегося в вершине стека сопроцессора – регистре st(0). Команда не имеет операндов. Результат выделения помещается в два регистра стека – мантисса в st(0), а порядок в st(1). При этом мантисса представляется вещественным числом с тем же знаком, что и у исходного числа, и порядком равным нулю. Порядок, помещенный в st(1), представляется как истинный порядок, то есть без константы смещения, в виде вещественного числа со знаком, и соответствует величине p формулы (1.3).

Команда масштабирования

fscale – эта команда изменяет порядок значения, находящегося в вершине стека сопроцессора – регистре st(0), на величину в st(1). Команда не имеет операндов. Величина в st(1) рассматривается как число со знаком. Его прибавление к полю порядка вещественного числа в st(0)означает его умножение на величину 2.

Сопроцессор имеет программно-аппаратные средства для округления тех результатов работы команд, которые не могут быть точно представлены. Но операция округления может быть проведена к значению в регистре st(0) и принудительно, для этого предназначена последняя команда в группе дополнительных команд – команда frndint. Эта команда округляет до целого значение, находящееся в вершине стека сопроцессора – регистре st(0). Команда не имеет операндов.

Возможны четыре режима округления величины в st(0), которые определяются значениями в двухразрядном поле rc управляющего регистра сопроцессора. С помощью двух команд fstcwr и fldcwr можно изменить режим округления. Эти команды, соответственно, записывают в память содержимое управляющего регистра сопроцессора и восстанавливают его обратно. Таким образом, пока содержимое это- го регистра находится в памяти, можно установить необходимое значение поля rc.

2.5 Команды трансцендентных функций

Сопроцессор имеет ряд команд, предназначенных для вычисления значений тригонометрических функций, таких как синус, косинус, тангенс, арктангенс, а также значений логарифмических и показательных функций.

Значения аргументов в командах, вычисляющих результат тригонометрических функций, должны задаваться в радианах.

Команды трансцендентных функций:

fcos – команда вычисляет косинус угла, находящийся в вершине стека сопроцессора – регистре st(0). Команда не имеет операндов. Результат возвращается в регистр st(0).

fsin – команда вычисляет синус угла, находящийся в вершине стека сопроцессора – регистре st(0). Команда не имеет операндов. Результат возвращается в регистр st(0).

fsincos – команда вычисляет синус и косинус угла, находящие ся в вершине стека сопроцессора – регистре st(0). Команда не имеет операндов. Результат возвращается в регистрах st(0) и st(1). При этом синус помещается в st(0), а косинус – в st(1).

fptan – команда вычисляет *частичный* тангенс угла, находя щийся в вершине стека сопроцессора – регистре st(0). Команда не имеет операндов. Результат возвращается в регистрах st(0) и st(1).

fpratan – команда вычисляет *частичный* арктангенс угла, находящийся в

вершине стека сопроцессора – регистре st(0). Команда не имеет операндов. Результат возвращается в регистрах st(0) и st(1)(угол от 0 до π / 4).

Отметим еще две команды сопроцессора: fpremи fprem1.

Команда fprem – команда получения *частичного* остатка от деления. Исходные значения делимого и делителя размещаются в стеке – делимое в st(0), делитель в st(1). Делитель рассматривается как некоторый *модуль.* Поэтому в результате работы команды получается остаток от деления по модулю. Но произойти это может не сразу, так как этот результат в общем случае достигается за несколько последовательных обращений к команде fprem, если значения операндов сильно различаются. Физически работа команды заключается в реализации хорошо известного всем действия: деления в столбик. При этом каждое промежуточное деление осуществляется отдельной командой fprem. Цикл, центральное место в котором занимает команда fprem, завершается, когда очередная полученная разность в st(0) становится меньше значения модуля в st(1). Судить об этом можно по состоянию флага c2в регистре состояния swr:

– если c2 = 0, то работа команды fprem полностью завершена, так как разность в st(0)меньше значения модуля в st(1);

– если c2= 1, то необходимо продолжить выполнение команды fprem, так как разность в st(0)больше значения модуля в st(1).

Таким образом, необходимо анализировать флаг c2 в теле цикла. Для этого c2 записывается в регистр флагов основного процессора с последующим анализом его командами условного перехода. Другой способ заключается в сравнении st(0) и st(1).

Важно отметить, то команда fprem не соответствует последнему стандарту IEEE-754 на вычисления с плавающей точкой. По этой причине в систему команд сопроцессора 1387 была введена команда fprem1, которая отличается от команды fprem тем, что накладывается дополнительное требование на значение остатка в st(0). Это значение не должно превышать половины модуля в st(1). В остальном работа команды fprem1аналогична работе fprem.

f2xm1 – команда вычисления значения функции $y = 2^{x-1}$. Исходное значение x размещается в вершине стека сопроцессора в регистре st(0)и должно лежать в диапазоне $-1 \le x \le 1$. Результат y замещает x в регистр st(0).

fyl2x – команда вычисления значения функции $z = y \log_2 (x)$. Исходное значение x размещается в вершине стека сопроцессора, а исходное значение y – в регистре st(1). Значение x должно лежать в диапазоне $0 < x < +\infty$, а значение y – в диапазоне $-\infty < y < +\infty$. Перед тем как осуществить запись результата z в вершину стека, команда fyl2x выталкивает значения x и y из стека и только после этого производит запись z в регистр st(0).

Команда fyl2xp1 – команда вычисления значения функции:

$$z = y \log_2 (x + 1).$$

Исходное значение x размещается в вершине стека со процессора – регистре st(0), а исходное значение y – в регистре st(1). Значение x должно лежать в диапазоне:

$$0 < |x| < 1 - \frac{1}{\sqrt{2}},$$

а значение y – в диапазоне $-\infty < y < +\infty$. Перед тем, как записать результат z в вершину стека – регистр st(0), команда fyl2xp1 выталкивает из стека значения x и y.

2.6 Команды управления сопроцессором

Эта группа команд предназначена для общего управления работой сопроцессора. Команды этой группы имеют особенность – перед началом своего выполнения они не проверяют наличие незамаскированных исключений. Однако такая проверка может понадобиться, в частности, для того, чтобы при параллельной работе основного процессора и сопроцессора предотвратить разрушение информации, необходимой для корректной обработки исключений, возникающих в сопроцессоре. Поэтому некоторые команды управления имеют аналоги, выполняющие те же действия плюс одну дополнительную функцию – проверку наличия исключения в сопроцессоре. Эти команды имеют одинаковые мнемокоды (и машинные коды тоже), различающиеся только вторым символом – символом n:

– мнемокод, не содержащий второго символа n, обозначает команду, которая перед началом своего выполнения проверяет наличие незамаскированных исключений;

– мнемокод, содержащий второй символ n, обозначает команду, которая перед началом своего выполнения не проверяет наличия незамаскированных исключений, то есть выполняется немедленно, что позволяет сэкономить несколько машинных тактов.

Эти команды имеют одинаковый машинный код. Отличие лишь в том, что перед командами, не содержащими символа n, транслятор ассемблера вставляет команду wait. Команда wait является полноценной командой основного процессора, и ее при необходимости можно указывать явно. Команда wait имеет аналог среди команд сопроцессора – fwait. Обеим этим командам соответствует код операции 9bh.

wait/fwait – команда ожидания. Она предназначена для синхронизации работы процессора и сопроцессора.

Команда инициализации сопроцессора finit/fninit. Она инициализирует управляющие регистры сопроцессора определенными значениями.

Следующие две команды работают с регистром состояния swr. fstsw/fnstsw ax – команда сохранения содержимого регистра состояния swr в регистре ax. Эту команду целесообразно использовать для подготовки к условным переходам по описанной при рассмотрении команд сравнения схеме.

fstsw/fnstsw приемник – команда сохранения содержимого регистра состояния swr в ячейке памяти. От рассмотренной ранее команда отличается типом операнда – теперь это ячейка памяти размером два байта (в соответствии с размерностью регистра swr).

Следующие две команды, работающие с информацией в регистре управления cwr, поддерживают действие записи и чтения содержимого этого регистра.

fstcw/fnstcw приемник – команда сохранения содержимого регистра управления cwr в ячейке памяти размером два байта. Эту команду целесообразно использовать для анализа полей маскирования исключений, управления точностью и округления. Следует заметить, что операндом не является регистр ax, в отличие от команды fstsw/fnstsw.

fldcw источник – команда загрузки значения ячейки памяти размером 16 битов в регистр управления cwr. Эта команда выполняет действие, противоположное действию команды fstcw/fnstcw. Команду fldcw целесообразно использовать для задания или изменения режима работы сопроцессора.

Команда без операндов fclex/fnclex позволяет сбросить флаги исключений в регистре состояния swr сопроцессора.

Сопроцессор имеет две команды, которые работают с указателем стека в регистре swr.

fincstp– команда увеличения указателя стека на единицу (поле

top) в регистре swr. Команда не имеет операндов.

fdecstp – команда уменьшения указателя стека (поле top) в регистре swr. Команда не имеет операндов.

ffree st(i) – команда помечает любой регистр стека сопроцессора как пустой. Это команда освобождения регистра стека st(i).

В группе команд управления можно выделить подгруппу команд, работающих с так называемой средой сопроцессора. *Среда сопроцессора* – это совокупность регистров сопроцессора и их значений. Среда сопроцессора может быть *частичной* или *полной.* О какой именно среде идет речь, определяется одной из рассмотренных далее команд:

fsave/fnsave приемник – команда сохранения полного состояния среды сопроцессора в память, адрес которой указан операндом приемник.

frstor источник – команда восстановления полного состояния среды сопроцессора из области памяти, адрес которой указан операндом источник. Сопроцессор будет работать в новой среде сразу после окончания работы команды frstor.

Пример вычисления выражения

$$y = a + bc - d / (a + b)$$

использованием целочисленных арифметических команд.

```
;Вычисление выражения y=a+b*c-d/(a+b)
;целочисленные арифметические команды сопроцессора
masm
model use16 small
.stack 100h
;сегмент данных
```

```
.data
a dw 2
b dw 3
c dw 4
d dw 5
z dw ?
y dw ?
  ;сегмент кода
.code main proc
mov ax, @data
mov ds, ax
finit
  ;приведение сопроцессора в начальное состояние
fild b      ;загрузка значение b в st(0)
fimul c   ;st(0)=bc
fiadd a   ;st(0)=bc+a
fistp z   ;z=bc+a
fild a ;загрузка значение a в st(0)
fiadd b   ;st(0)=b+a
fidivr d ;st(0)=d/(b+a)
fisubr z ;z=st(0)-z
fistp y ;результат в y
mov ax,4c00h
int 21h
main endp
end main
```

Пример вычисления выражения

$$y = a + bc - d / (a + b)$$

с использованием вещественных арифметических команд.

```
;Вычисление выражения y=a+b*c-d/(a+b).
;Вещественные арифметические команды сопроцессора
.masm
model use16 small
.stack 100h
.data ;сегмент данных
a dd 2.5
b dd 3.5
c dd 4.5
```

```
d dd 5.5z   d
d   ?
y dd ?
.code
main proc
mov ax,@data
mov ds, ax
finit
;приведение сопроцессора в начальное состояние
fld b        ;загрузка значение b в st(0)
fld c        ;загрузка значение c в st(0) st(1)=bfmul   ;st(0)=bc
fadd a       ;st(0)=bc+afstp z ;z=bc+a
fld a        ;загрузка значение a в st(0)fadd b  ;st(0)=b+a
fld d        ;st(0)=d st(1)=b+afdiv   ;st(0)=d/(b+a)
fld z        ;st(0)=a+bc st(1)=d/(b+a)
fxchst(1) ;st(0) и st(1) меняем местамиfsub   ;st(0)=st(0)-st(1)
fstp y       ;результат в ymov ax,4c00h
int 21h
main endp
end main
```

Пример вычисления выражения $y = \begin{cases} ab, & a+b<5; \\ 2a-b, & a+b>5; \\ a/b, & a+b=5; \end{cases}$

с использованием целочисленных арифметических команд. Программа написана с использованием внутренних процедур (каждая «ветка» вычисляется в отдельной процедуре).

При вычислении значения этой функции необходимо анализировать условие. В таблице 2.1 представлены значения флагов и команды условного перехода, которые используются для проверки условий в регистрах сопроцессора.

Таблица 2.1 – Значения флагов и используемые команды условного перехода при работе с регистрами сопроцессора

Условия	c3	c2	c0	Команда условного перехода
st(0) > операнда	0	0	0	jnc
st(0) < операнда	0	0	1	jc
st(0) = операнда	1	0	0	jz
Неупорядоченная пара	1	1	1	
Флаги	zf	pf	cf	

Команды условного перехода:

```
jc metka    ;переход, если cf=1
jnc metka   ;переход, если cf=0
jp metka    ;переход, если pf=1
jnp metka   ;переход, если pf=0
jz metka    ;переход, если zf=1
jnz metka ;переход, если zf=0
```

```
;Вычисление выражения y=a·b, если a+b<5,
;                              y=2·a–b, если a+b>5,
;                              y=a/b, если a+b=5.
;Целочисленные арифметические команды сопроцессора
.masm
model use16 small
.486
.stack 100h
.data ;сегмент данных
a dw 4
b dw 2
y dw ?
five dw 5
two   dw 2
.code
main proc
mov ax,@data
mov ds,ax
finit
;приведение сопроцессора в начальное состояние
fild a      ;загрузка значение a в st(0)
fiadd b   ;st(0)=a+b
ficomp five ;сравниваем st(0) с 5
```

```
;и одновременно сбрасываем регистр st(0)
fstsw ax ;сохранение swr в регистре ax
sahf        ;запись swr->ax-> регистр флагов
jc met1   ;если a+b<5 переход на метку1
jz met2   ;если a+b=5 переход на метку2
call p2   ;вычисляем значение при a+b>5
 jmp exit
met1: call p1   ;вычисляем значение при a+b<5
jmp exit
met2: call p3   ;вычисляем значение при a+b=5
exit:
mov ax,4c00h
int 21h
main endp
p1 proc  ;вычисляем значение при a+b<5
fild a       ;st(0)=a
fimul b ;st(0)=a*b
fist y
ret
p1 endp
p2 proc    ;вычисляем значение при a+b>5
fild a         ;st(0)=a
fimul two ;st(0)=2*a
fisub b      ;st(0)=2*a-b
fist y         ;y=2*a-b
ret p2 endp
p3 proc ;вычисляем значение при a+b=5
fild a       ;st(0)=a
fidiv b ;st(0)=a/b
fist y
ret
p3 endp
end main
```

Пример вычисления выражения $y = \begin{cases} ab, & a+b<5; \\ 2a-b, & a+b>5; \\ a/b, & a+b=5; \end{cases}$

с использованием вещественных арифметических команд. Программу написана с использованием внутренних процедур (каждая «ветка» вычисляется в отдельной процедуре).

```
;Вычисление выражения y=a*b,          если a+b<5,
;                                     y=2*a-b, если a+b>5,
;                                     y=a/b,          если a+b=5.
;Вещественные арифметические команды сопроцессора.masm
model use16 small
.486
.stack 100h
.data ;сегмент данных
a dd 4.5
b dd 2.5
y dd ?
five dd 5.0
two   dd 2.0
.code main proc
mov ax, @dat
amov ds, ax
finit ;приведение сопроцессора в начальное состояние
fld a ;загрузка значение a в st(0)
fadd b ;st(0)=a+b
fcomp five
  ;сравниваем st(0) с 5 и одновременно сбрасываем
  ;регистр st(0)
fstsw ax ;сохранение swr в регистре ax
sahf       ;запись swr->ax-> регистр флагов
jc met1   ;если a+b<5 переход на метку1
jz met2   ;если a+b=5 переход на метку2
call p2   ;вычисляем значение при a+b>5
jmp exit
met1: call p1 ;вычисляем значение при a+b<5
```

```
jmp exit
met2: call p3 ;вычисляем значение при a+b=5
exit:
mov ax, 4c00h
int 21h
main endp
p1 proc         ;вычисляем значение при a+b<5
fld a           ;st(0)=a
fmul b          ;st(0)=a*b
fst y
ret
p1 endp
p2 proc;вычисляем значение при a+b>5
fld a     ;st(0)=a
fmul two ;st(0)=2*a
fsub b ;st(0)=2*a-b
fst y ;y=2*a-b
p2 endp
p3 proc ;вычисляем значение при a+b=5
fld a ;st(0)=a
fst y ret
p3 endp
end main
```

3 Лабораторная работа №1. Архитектура и программирование сопроцессора. Использование целочисленных команд

Цель работы: изучение архитектуры и программной модели сопроцессора; практическая работа с системой целочисленных команд

1. Изучить следующие теоретические сведения: архитектура и программная модель сопроцессора – регистры и их назначение; форматы чисел, типовые команды, целочисленные команды (используя материалы лекций и методические указания).

2. Составить и отладить программу на языке ассемблера для вычисления значения функции, используя регистры сопроцессора и целочисленные команды согласно варианту, выданному преподавателем (таблица 2.2). Оформить ее в виде отдельной функции. Каждую ветку алгоритма оформить в виде отдельной внутренней процедуры. Ввод и вывод данных осуществляется через консоль. Программу протестировать по всем условиям.

3. Подготовиться к защите

Содержание отчета:

- титульный лист;
- описание варианта задания;
- текст программы;
- скриншоты результатов.

Таблица 2.2 – Варианты заданий по системе команд сопроцессора

Вариант 1 $y=\begin{cases} 6xy-4y, & x+y>9; \\ \dfrac{2xy+3+x}{x^2+3y^2+1}, & x+y<-1; \\ 3x^2-2y+6, & -1\le x+y\le 9. \end{cases}$	**Вариант 2** $y=\begin{cases} 4xy+5, & x+y>10; \\ \dfrac{3xy+2y}{x^2+y^2+1}, & x+y<-2; \\ 6x-2y^2+1, & -2\le x+y\le 1. \end{cases}$
Вариант 3 $y=\begin{cases} 4xy+4x, & xy>8; \\ \dfrac{2xy+3}{x^2+2y^2-1}, & xy<-12; \\ 3x^2-2y^2+1, & -12\le xy\le 8. \end{cases}$	**Вариант 4** $y=\begin{cases} 2xy+3, & x-y>9; \\ \dfrac{3y^2}{x^2+2y+1}, & x-y<-6; \\ 2x^2-2y+1, & -6\le x-y\le 9. \end{cases}$
Вариант 5 $y=\begin{cases} 3x+2y^2-5, & xy>8; \\ \dfrac{xy+7}{2x^2+4}, & xy<-12; \\ 4x^2-3y^2+4, & -12\le xy\le 8. \end{cases}$	**Вариант 6** $y=\begin{cases} 5x+2xy+1, & x+y>9; \\ \dfrac{2xy+3}{y^2+4}, & x+y<-5; \\ 3x^2-2y^2+1, & -5\le x+y\le 9. \end{cases}$

Вариант 7

$$y=\begin{cases}3x+2y^2-1, & x+y>8;\\ \dfrac{2xy+3}{y^2+2x+1}, & x+y<-5;\\ 2x^2-3y^2+2, & -5\le x+y\le 8.\end{cases}$$

Вариант 8

$$y=\begin{cases}3x+4y+3, & x+y>7;\\ \dfrac{3y-2x^2}{y^2+4}, & x+y<-5;\\ 3x^2-4y^2-3, & -5\le x+y\le 7.\end{cases}$$

Вариант 9

$$y=\begin{cases}\dfrac{x^2+2y}{x^2+y^2+1}, & x-y>15;\\ \dfrac{y^2-x}{x^2+y^2-1}, & x-y<-5;\\ x^2+2y^2+4, & -5\le x-y\le 15.\end{cases}$$

Вариант 10

$$y=\begin{cases}\dfrac{x^2+3y}{x^2+2y^2+1}, & x+y>6;\\ \dfrac{3y^2-1}{2x^2+y^2+1}, & x+y<-5;\\ 2x^2+3y^2-4, & -5\le x+y\le 6.\end{cases}$$

Вариант 11

$$y=\begin{cases}\dfrac{y+2x}{x^2+3y^2+1}, & x+y>11;\\ \dfrac{4xy}{x^2+2y^2+1}, & x+y<-4;\\ 5x^2+2y^2, & -4\le x+y\le 11.\end{cases}$$

Вариант 12

$$y=\begin{cases}\dfrac{x^2-4y+5}{x^2+2}, & x-y>2;\\ \dfrac{2x+y}{3x^2+4}, & x-y<-4;\\ x^2-2y+3, & -4\le x-y\le 2.\end{cases}$$

Вариант 13

$$y=\begin{cases}\dfrac{2x+y+4}{xy^2+5}, & x-y>8;\\ 2+3x^2+y, & x-y<-2;\\ 5x^2-2y, & -2\le x-y\le 8.\end{cases}$$

Вариант 14

$$y=\begin{cases}\dfrac{1+x}{5x^2-y^2+1}, & x-y<-4;\\ 3x+5y^2, & x-y>6;\\ x^3-2y, & -4\le x-y\le 6.\end{cases}$$

Вариант 15

$$y=\begin{cases}5xy-3, & x+y>8;\\ \dfrac{4xy^2}{x^2+1}, & x+y<-5;\\ 2x^2-4y^2+3, & -5\le x+y\le 8.\end{cases}$$

Вариант 16

$$y=\begin{cases}x^2-2y, & x+y\le -3;\\ \dfrac{xy+5}{x^2+y^2+1}, & -3<x+y<5;\\ 2xy-3y^2, & x+y\ge 5.\end{cases}$$

Вариант 17

$$y=\begin{cases}4x+y-3, & x+y>2;\\ \dfrac{3xy+2y}{x^2+2y^2+1}, & x+y<-10;\\ 5x-3y^2+7, & -10\le x+y\le 2.\end{cases}$$

Вариант 18

$$y=\begin{cases}5xy+4y, & x+y>9;\\ \dfrac{xy-5}{x^2+2y^2-1}, & x+y<-1;\\ 3x^2-2y+6x, & -1\le x+y\le 9.\end{cases}$$

4 Лабораторная работа №2. Архитектура и программирование сопроцессора. Использование вещественных команд

Цель работы: изучение архитектуры и программной модели сопроцессора; практическая работа с системой вещественных команд

1. Изучить систему вещественных команд сопроцессора (используя материалы лекций и методические указания).
2. Составить и отладить программу на языке ассемблера для вычисления значения функции, используя регистры сопроцессора и вещественные команды согласно варианту, выданному преподавателем (таблица 2.2). Оформить ее в виде отдельной функции. Каждую ветку алгоритма оформить в виде отдельной внутренней процедуры. Ввод и вывод данных осуществляется через консоль. Программу протестировать по всем условиям.
3. Подготовиться к защите

Содержание отчета:

- титульный лист;
- описание варианта задания;
- текст программы;
- скриншоты результатов.

Список использованных источников

1. Юров В. И. Assembler: Учебник / В. И. Юров. – 2-е изд. – Санкт-Петербург: Питер, 2011. – 637 с.
2. Ружицкая, Е. А. Программирование на языке Assembler : архитектура и программирование сопроцессора : практическое пособие / Е. А. Ружицкая; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2016. – 46 с.
3. Ружицкая, Е. А. Программирование на языке Assembler : системы счисления, программная модель микропроцессора, арифметические команды: практическое пособие / Е. А. Ружицкая, О. Г. Осипова, В. А. Ковалёва ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2016. – 47 с.
4. Ружицкая, Е. А. Программирование на языке Assembler : программирование ветвящихся и циклических вычислительных процессов, обработка массивов : практическое пособие / Е. А. Ружицкая, О. Г. Осипова, В. А. Ковалёва ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2016. – 47 с.

2.3 ЛАБОРАТОРНЫЙ ПРАКТИКУМ. ЧАСТЬ 3. АНАЛИЗ СЕТЕВОГО ТРАФИКА И ПРОТОКОЛОВ (НА БАЗЕ WIRESHARK)

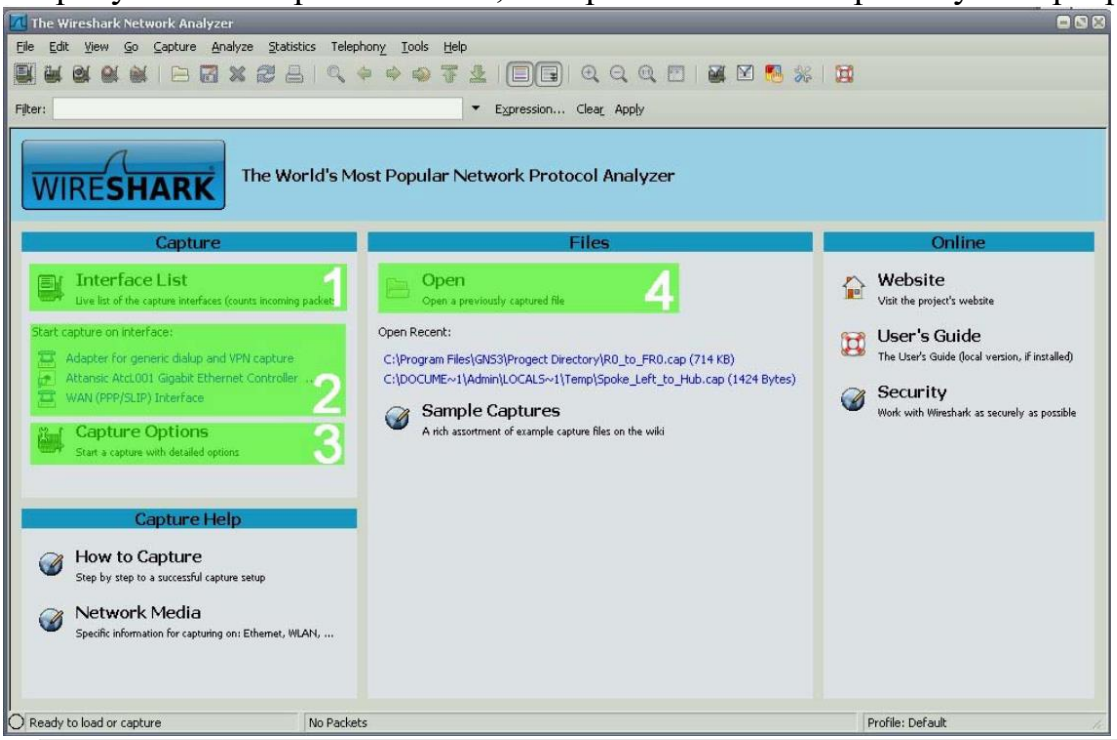
ОГЛАВЛЕНИЕ

1 Краткие теоретические сведения

Wireshark - это программный анализатор трафика, который позволяет перехватывать информационные потоки, передаваемые по сети. Программа в первую очередь предназначена для сбора информации о сетевых взаимодействиях и для обнаружения и устранения неполадок в сети. Анализаторы трафика (сниферы) так же часто применяются при разработке новых протоколов и программного обеспечения и в образовательных целях.

Установленная и запущенная на компьютере программа Wireshark позволяет обнаружить и изучить любой протокольный блок данный (Protocol Data Unit, PDU), который был отправлен или получен с помощью любого из установленных на компьютере сетевых адаптеров (Network Interface Card, NIC). Начальная настройка программы и запуск захвата трафика.

На рисунке 1 изображено окно, которое появляется при запуске программы



Выделенная область	Описание и функции
1	Кнопка, при нажатии на которую программа выведет список активных сетевых адаптеров (рисунок X), с которых возможен захват трафика. Список имеет вид интерактивной таблицы.
2	Список активных сетевых интерфейсов. Нажатие на любой интерфейс из списка немедленно запустит процесс захвата трафика.
3	Кнопка, при нажатии на которую программа выведет окно настроек процесса захвата трафика (рисунок X).
4	Кнопка, позволяющая загружать в программу захваченный ранее и сохраненный файл и отчётом о захваченном сетевом трафике.

Рисунок 1 – Стартовый интерфейс программы

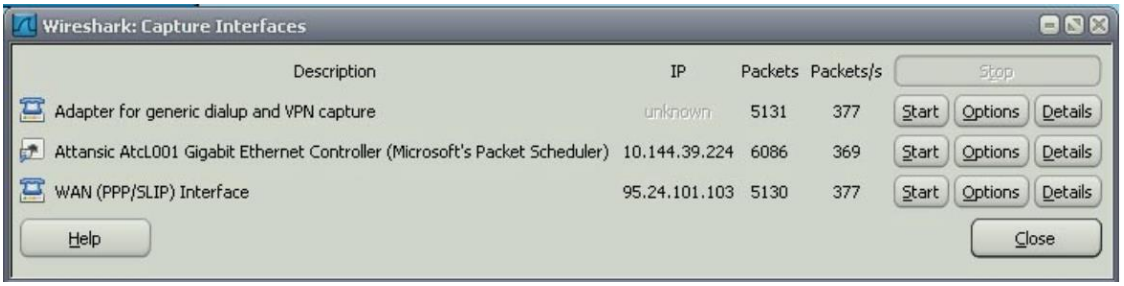


Рисунок 2 – Список активных сетевых адаптеров

Список активных адаптеров имеет вид интерактивной таблицы со следующими полями:

Поле таблицы	Описание
Description	Описание адаптера
IP	Сетевой адрес (Если есть)
Packets	Количество захваченных блоков данных (PDU) с момента вызова таблицы.
Packets/s	Скорость обработки (приёма и отправки пакетов).

Также напротив каждого интерфейса расположены 3 кнопки (рисунок 3, таблица 1)

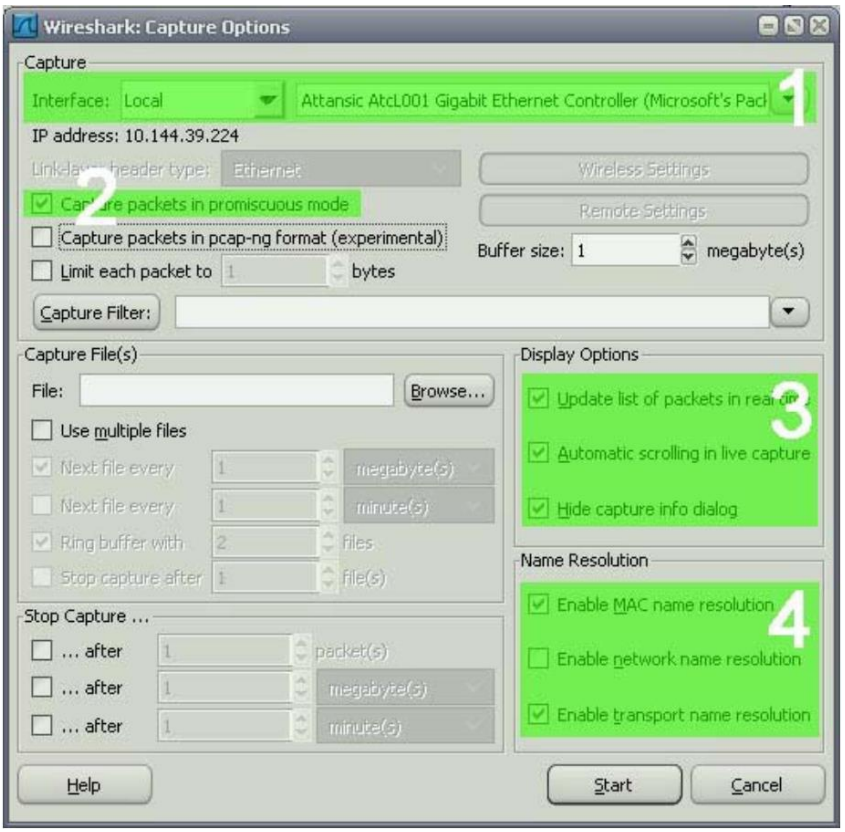


Рисунок 3 – Окно настроек захвата сетевого трафика

Таблица 1 – Окно настроек захвата сетевого трафика

Выделенная область	Описание и функции
1	Выбор интерфейса для захвата трафика. В этой области расположены два выпадающих меню. Первое (левое) определяет тип используемого интерфейса: локальный (Local) или удалённый (Remote). Второе (правое) выпадающее меню определяет сам интерфейс.
2	**Capture packets in promiscuous mode –** Захват пакетов в режиме приёма всех сетевых пакетов. Если эта опция включена, программа будет захватывать все PDU, которые принимает сетевой адаптер. Если опция отключена – программа будет захватывать только PDU, предназначенные компьютеру, на котором она установлена.
3	Опции отображения захвата пакетов: **Update list in real time –** обновление списка в реальном времени. Если эта опция включена, то программа отображает захваченный трафик в реальном времени. **Automatic scrolling in live capture –** Автоматическая прокрутка при захвате. Если эта опция включена, программа будет автоматически удерживать в окне вывода захваченной информации последние захваченные PDU. **Hide capture info dialog –** Скрыть информационно-диалоговое окно захвата. Если эта опция включена, то информационно-диалоговое окно захвата (Рисунок X) не выводится.
4	Опции преобразования имен. **Enable MAC name resolution –** Включить преобразование MAC-адресов. Эта опция включает автоматическое преобразование физических адресов устройств в более понятный для человека формат. Пример: **00:09:5b:**01:02:03 -> **Netgear_**01:02:03. Выделенная часть сетевого адреса закреплена за производителем **Netgear,** поэтому программа преобразовала эту часть в название производителя. Примечание: если включена опция преобразования сетевых имён, то в некоторых случаях программа выводит DNS имя вместо MAC-адреса. **Enable network name resolution –** Включить преобразование сетевых имён. Эта опция включает автоматическое преобразование сетевых адресов устройств в DNS имена устройств. Пример: **216.239.37.99** -> **www.google.com.** **Enable transport name resolution –** Включить преобразование TCP/UDP портов. Эта опция включает автоматическое преобразование TCP/UDP закреплённых за определёнными протоколами портов в названия этих протоколов. Пример: **80** -> **http**

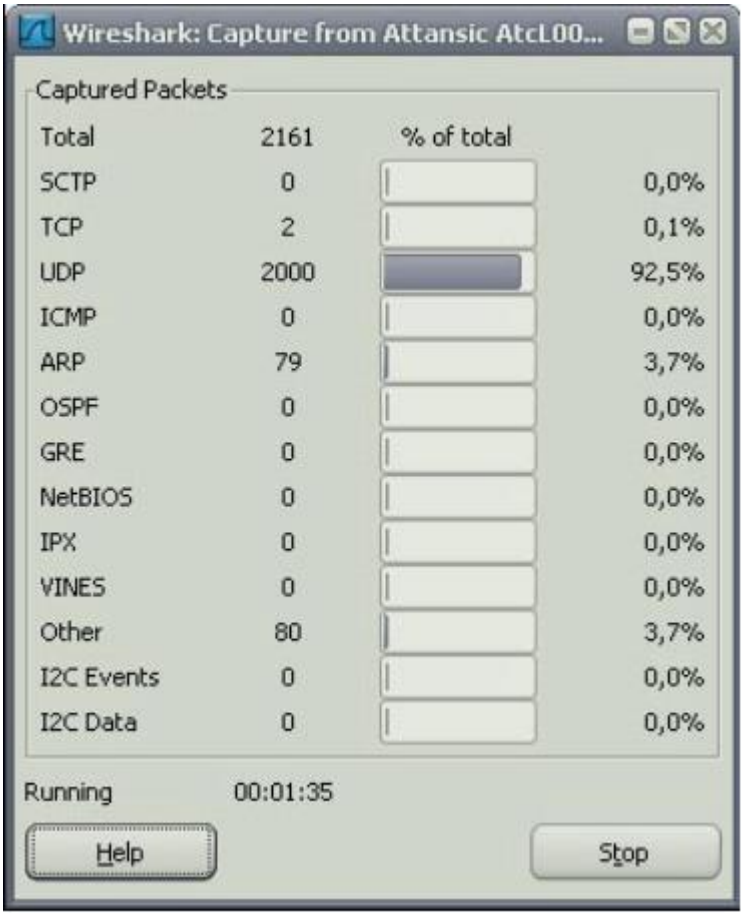


Рисунок 4 – Информационно-диалоговое окно захвата.

Список активных адаптеров имеет вид интерактивной таблицы со следующими столбцами – таблица 2.

Таблица 2 – Список активных адаптеров

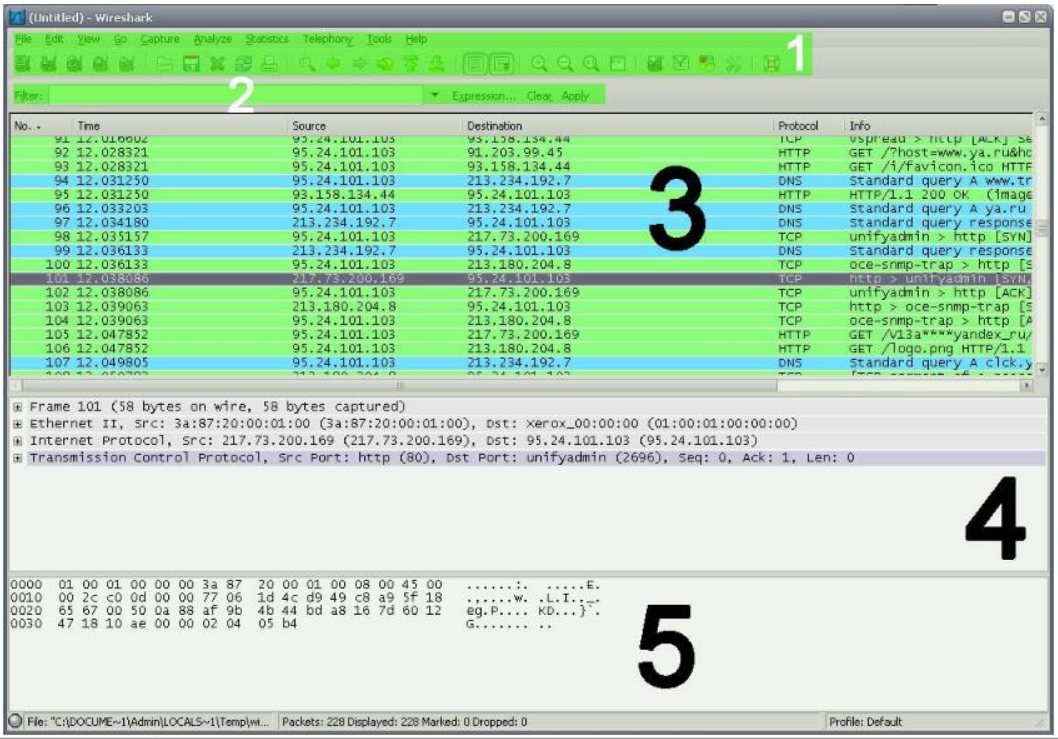
№ столбца (слева - направо)	Описание
1	Имя протокола. В таблице представлены наиболее распространенные протоколы.
2	Количество захваченных PDU определённого протокола.
3, 4	Графическое и числовое отображение процентного отношения захваченных PDU конкретного протокола к общему числу захваченных PDU.

Также в окне отображаются следующие параметры:

Параметр	Описание
Total	Общее количество захваченных пакетов.
Running	Время, на протяжении которого ведётся захват пакетов.

1.1 Главное рабочее окно программы

После выбора интерфейса и запуска захвата PDU программа вызовет окно, показанное на рисунке 5.



Выделенная область	Описание и функции
1	Меню программы, и панель инструментов, предоставляющая доступ к наиболее часто используемым функциям программы.
2	Фильтр, позволяющий производить выборочный захват PDU.
3	Поле списка PDU, в котором отображается краткая информация по всем захваченным PDU.
4	Информационное поле, в котором отображается подробная информация по выбранному PDU.
5	Поле, в котором отображаются данные выделенные в информационном поле в шестнадцатеричной и текстовой форме.

Рисунок 5 – Окно отображения захваченного трафика

1.2 Панель инструментов

Панель инструментов представлена на рисунке 6 и рисунке 7.

Рисунок 6 – Панель инструментов

№	Кнопка	Название кнопки	Соответствующая опция в меню	Функции кнопки
1		Interfaces…	Capture / Interfaces…	Вызов списка активных сетевых адаптеров (Рисунок X).
2		Options…	Capture / Options…	Вызов окна настроек захвата сетевого трафика (Рисунок X).
3		Start…	Capture / Start…	Старт захвата трафика с текущими параметрами захвата.
4		Stop…	Capture / Stop…	Остановить захват трафика.
5		Restart…	Capture / Restart…	Перезапустить захват трафика с текущими параметрами.
6		Open…	File / Open…	Открыть файл с отчётом о захваченном трафике.
7		Save As…	File / Save As…	Сохранить текущий отчёт о захваченном трафике в файл.
8		Close…	File / Close…	Закрыть текущий отчёт о захваченном трафике.
9		Reload…	View / Reload…	Закрыть и открыть заново текущий отчёт о захваченном трафике.
10		Print…	File / Print…	Распечатать текущий отчёт о захваченном трафике.
11		Zoom In…	View / Zoom In…	Увеличить размер шрифта.
12		Zoom Out…	View / Zoom Out…	Уменьшить размер шрифта.
13		Normal Size…	View / Normal Size…	Установить размер шрифта, используемый по умолчанию.
14		Preferences…	Edit / Preferences…	Вызов меню настроек.
15		Help…	Help / Contents…	Вызов справки.

Рисунок 7 – Панель инструментов

1.3 Фильтр

Фильтр позволяет настроить программу Wireshark на отображение только определённого, удовлетворяющего условиям текущего примененного фильтра сетевого трафика.

Фильтр может применяться как при захвате трафика в реальном времени, так и при анализе захвата, сохранённого в файле.

Панель фильтра представлена на рисунке 8 и рисунке 9.

Рисунок 8 – Панель фильтра

№	Кнопка / Поле	Название Кнопки / поля	Функции кнопки / поля
1	Filter:	Filter:	Вызов диалогового окна для создания и сохранения пользовательских фильтров (Рисунок X).
2		Filter Input	Поле ввода фильтра.
3			Вызов списка применённых ранее фильтров.
4	Expression...	Expression...	Вызов диалогового окна, позволяющего выбирать фильтры из базы данных программы.
5	Clear	Clear	Очистить поле ввода фильтра.
6	Apply	Apply	Применить фильтр.

Рисунок 9 – Панель фильтра

Для применения фильтра необходимо:

- ввести фильтр в поле ввода;
- нажать кнопку “Apply”.

Если фильтр введён в соответствии с правилами построения фильтров, то цвет поля ввода будет зелёным (рисунок 10), если фильтр введён с ошибкой – красным (рисунок 11).

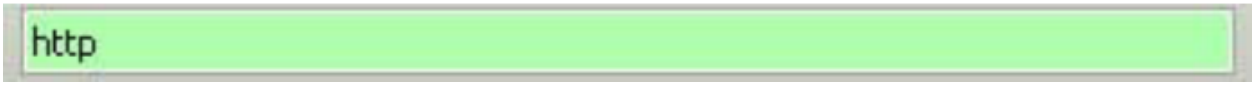


Рисунок 10 – Фильтр введён правильно

Рисунок 11 – Фильтр введён неправильно

Построение фильтров

Фильтрацию, применяемую в программе Wireshark можно условно разделить на две категории:

- фильтрация по определённым протоколам;
- фильтрация по определённым значениям полей в заголовках протоколов.

Для применения фильтрации по определённому протоколу необходимо ввести имя протокола в поле ввода фильтра. Пример выполнения фильтрации по протоколу HTTP показан на рисунках 12, 13.

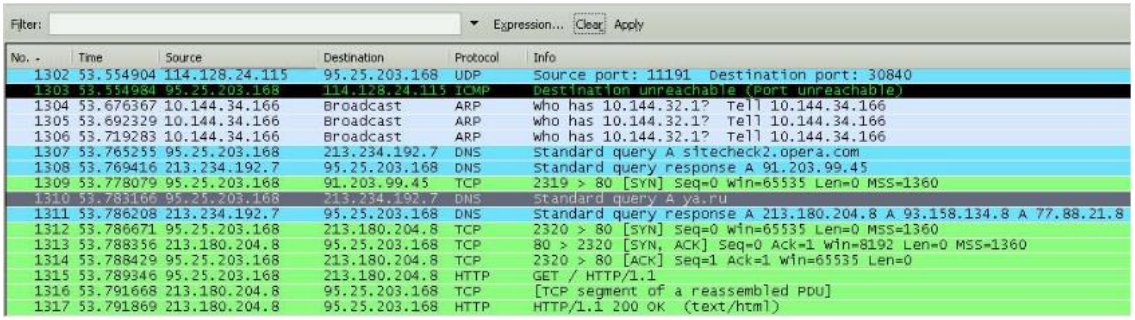


Рисунок 12 – Вывод программы до применения фильтра

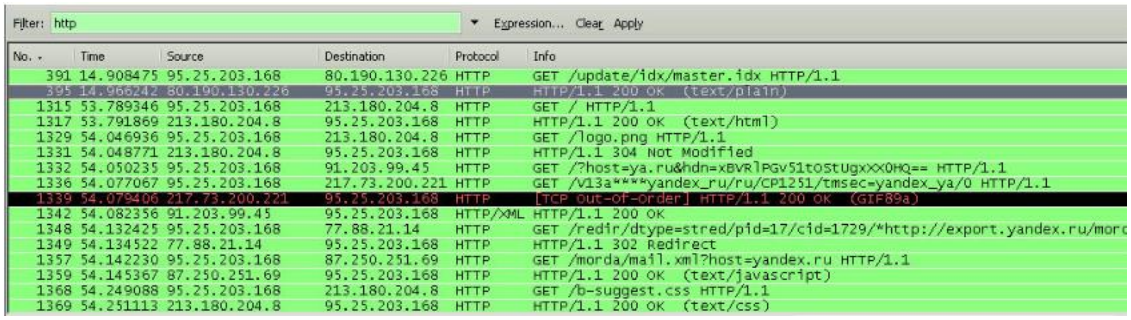


Рисунок 13 – Вывод программы после применения фильтра

Фильтрация по определённому значению поля в заголовках протоколов строится по следующему синтаксису:

Поле Оператор сравнения Значение

Операторы сравнения и некоторые обозначения полей, которые могут использоваться при построении фильтров, представлены в таблицах 3 и 4.

Таблица 3 – Обозначения полей при построении фильтров

Поле	Описание
eth.addr	Физический адрес источника или получателя в кадре протокола Ethernet.
eth.dst	Физический адрес получателя в кадре протокола Ethernet.
eth.src	Физический адрес источника в кадре протокола Ethernet.
eth.len	Длина кадра протокола Ethernet.
ip.addr	Сетевой адрес источника или получателя в пакете протокола IP.
ip.dst	Сетевой адрес получателя в пакете протокола IP.
ip.src	Сетевой адрес источника в пакете протокола IP.
ip.proto	Обозначения протокола, который был инкапсулирован в пакет IP.
tcp.ack	Подтверждения (ACK) протокола TCP
tcp.port	Порт источника или получателя в сегменте протокола TCP.
tcp.dstport	Порт получателя в сегменте протокола TCP.
tcp.srcport	Порт источника в сегменте протокола TCP.
udp.port	Порт источника или получателя в сегменте протокола UCP.
udp.dstport	Порт получателя в сегменте протокола UCP.
udp.srcport	Порт источника в сегменте протокола UCP.
dns.qry.name	Имя сетевого ресурса в DNS запросе.
dns.resp.name	Имя сетевого ресурса в DNS ответе.

Таблица 4 – Операторы сравнения

Оператор		Значение	Примеры
==	eq	Равно	**p.addr==192.168.1.1** Отображать только те пакеты протокола IP, в которых сетевой адрес отправителя или получателя равен 192.168.1.1 **eth.dst==ff:ff:ff:ff:ff:ff** Отображать только широковещательные (broadcast) кадры протокола Ethernet.
!=	ne	Не равно	**ip.dst==255.255.255.255** Не отображать широковещательные (broadcast) пакеты протокола IP.
>	gt	Больше	**tcp.dstport>10000** Отображать только те сегменты протокола TCP, в которых порт получателя больше 10000.
<	lt	Меньше	**tcp.dstport<1024** Отображать только те датаграммы протокола UDP, в которых порт получателя меньше 1024.

При построении фильтра можно комбинировать два и более условия, используя логические операторы.

Комбинирование условий при построении операторов производится по следующему принципу:

Условие 1 Логический оператор Условие 2 Логический оператор

В качестве условия может использоваться как фильтрация по протоколам, так и фильтрация по значениям определённых полей в протоколах.

В таблице 5 представлены некоторые логические операторы.

Таблица 5 – Некоторые логические операторы

Оператор		Значение	Примеры
&&	**and**	И	**ip.src==192.168.1.1 && ip.dst==192.168.1.10** Отображать только сообщения отправленные устройством с сетевым адресом 192.168.1.1 для устройства с сетевым адресом 192.168.1.10
||	**or**	ИЛИ	**eth.dst==ff:ff:ff:ff:ff:ff || ip.dst==255.255.255.255** Отображать только широковещательные кадры протокола Ethernet или пакеты протокола IP.
!	**not**	НЕ (Отрицание)	**!arp** Не отображать PDU протокола ARP.

Поле списка захваченных PDU. В поле списка захваченных PDU (Рисунок 14) выводится сводная информация по всему трафику, захваченному с помощью программы Wireshark.

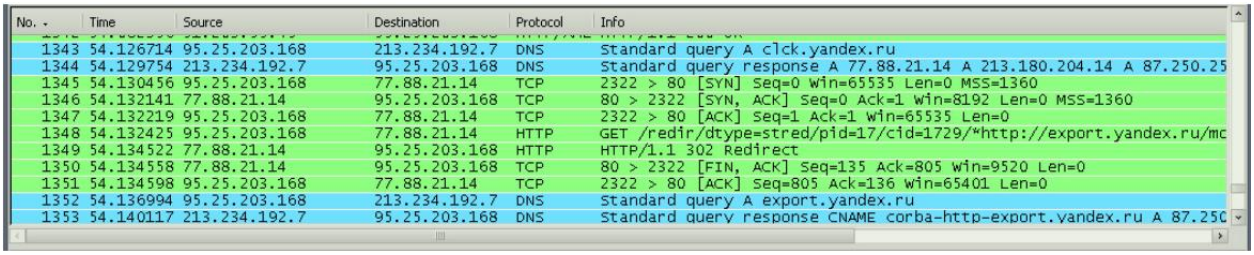


Рисунок 14 – После списка захваченных PDU

Сводная информация выводится в виде таблицы со следующими полями:

Поле таблицы	Описание
No.	Порядковый номер захваченного PDU. При использовании фильтра порядковый номер не изменяется.
Time	Временная отметка, обозначающая время (в секундах) прошедшее с момента начала захвата PDU.
Source	Сетевой адрес отправителя.
Destination	Сетевой адрес получателя.
Protocol	Протокол.
Info	Дополнительная информация о захваченном PDU.

На рисунке 15 представлен пример сводной информации о захваченной PDU

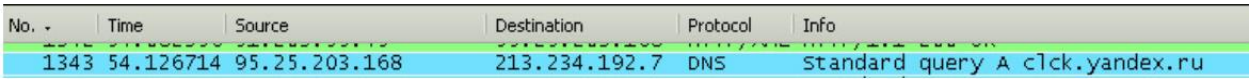


Рисунок 15 – Пример записи в списке захваченных PDU

Запись можно интерпретировать следующим образом:

1343 – этот PDU является 1343-им по счету захваченным PDU.

54.126714 – PDU захвачен через 54 секунды после начала захвата.

95.25.203.168 – Устройство, которое его отправило, имеет сетевой адрес 95.25.203.168.

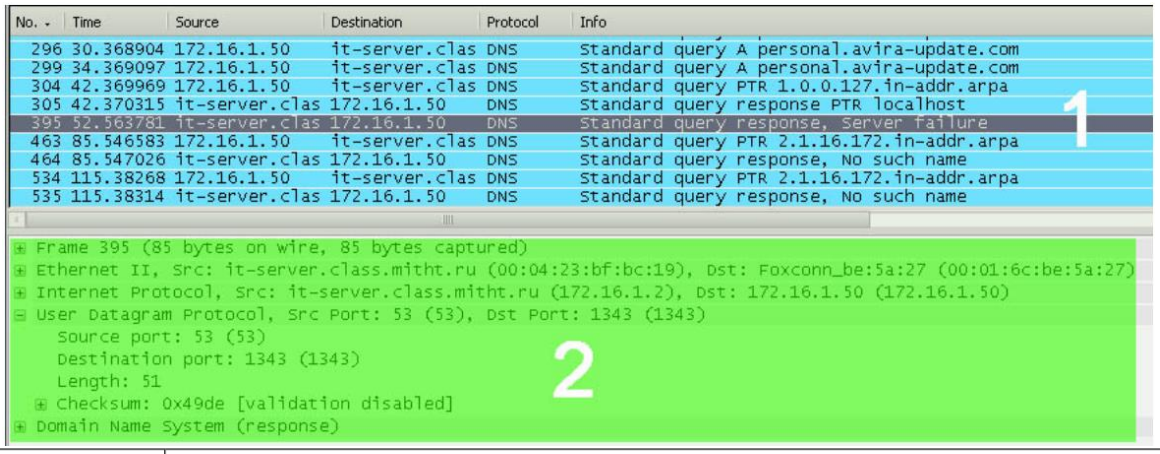
213.234.192.7 – Устройство, которому оно предназначалось, имеет адрес 213.234.192.7.

DNS – Взаимодействие между устройствами происходит по протоколу DNS.

Standard query A click.yandex.ru – устройство с адресом 95.25.203.168 обращается к устройству с адресом 213.234.192.7, чтобы узнать сетевой адрес информационного ресурса click.yandex.ru

1.4 Информационное поле

В информационном поле (Рисунок 14) отображается подробная информация о захваченном PDU, выделенном в поле списка захваченных PDU.



Выделенная область	Описание и функции
1	Выделенная запись в листе списка захваченных PDU. Запись выделятся нажатием левой кнопки мыши. Программа помечает текущую выделенную запись серым цветом.
2	Подробная информация о выделенном PDU.

Рисунок 14 – Информационное поле программы

Информация о выделенном PDU выводится в виде иерархического списка. Иерархия списка соответствует порядку инкапсуляции данных, применяемой при использовании протоколов стека TCP/IP для передачи информации между устройствами. На рисунке 15 показан пример вывода информации о захваченном PDU протокола HTTP.

Интерпретация вложенных списков. Каждый вложенный список представляет собой последовательность полей (всех, или основных), содержащихся в заголовке протокола, используемого при инкапсуляции данных.

Порядок полей в списке соответствует порядку полей в заголовке протокола. Ниже рассмотрены некоторые распространённые протоколы и интерпретация полей их заголовков.

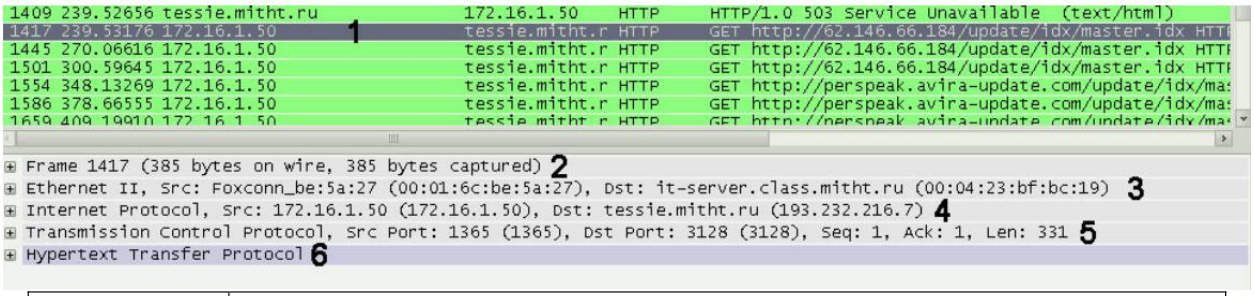
1.5 Протокол Ethernet

Стандарты Ethernet определяют проводные соединения и электрические сигналы на физическом уровне, формат кадров и протоколы управления доступом к среде — на канальном уровне модели OSI.

Схематичное изображение кадра протокола Ethernet и соответствующий вывод программы Wireshark показаны на рисунке 16.

Зелёным цветом выделены поля, выводимые программой.

Информацию в заголовке списка можно интерпретировать следующим образом:



Выделенная область	Описание и функции
1	Выделенное PDU в поле списка захваченных PDU. В соответствии с установками по умолчанию, программа отмечает выделенное PDU серым цветом.
2	⊞ Frame 1417 В этом вложенном списке содержится справочная информация о захваченном PDU, такая как: время захвата, длинна PDU и т.д.
3	⊞ Ethernet II, В этом вложенном списке расположена информация о заголовке протокола канального (Data Link) уровня. В данном случае это протокол Ethernet.
4	⊞ Internet Protocol, В этом вложенном списке расположена информация о заголовке протокола сетевого (Network) уровня. В данном случае это протокол IP.
5	⊞ Transmission Control Protocol, В этом вложенном списке расположена информация о заголовке протокола транспортного (Transport) уровня. В данном случае это протокол TCP.
6	⊞ Hypertext Transfer Protocol В этом вложенном списке расположена информация о заголовке протокола транспортного (Application) уровня. В данном случае это протокол HTTP.

Рисунок 15 – Информация о захваченном PDU протокола HTTP

Ethernet II – это кадр протокола Ethernet;

Src: Foxconn_be:5a:27 (00:01:6c:be:5a:27) – физический адрес устройства отправителя, 00:01:6c:be:5a:27, производитель сетевой карты – компания Foxconn;

Dst: it-server.class.mitht.ru (00:04:23:bf:bc:19) – физический адрес устройства получателя 00:04:23:bf:bc:19, DNS имя устройства - it-server.class.mitht.ru.

1.6 Протокол IP

Протокол IP — протокол сетевого уровня, обеспечивающий систему глобальной логической адресации для устройств в сети.

Схематичное изображение заголовка пакета протокола IP и соответствующий вывод программы Wireshark показаны на рисунке 17.

Зелёным цветом выделены поля, выводимые программой.

7	1	6	6	2	46-1500	4
Preamble	Start of Frame	**Destination**	**Source**	**Type**	Data	FCS

```
⊞ Frame 1417 (385 bytes on wire, 385 bytes captured)
⊟ Ethernet II, Src: Foxconn_be:5a:27 (00:01:6c:be:5a:27), Dst: it-server.class.mitht.ru (00:04:23:bf:bc:19)
  ⊞ Destination: it-server.class.mitht.ru (00:04:23:bf:bc:19)
  ⊞ Source: Foxconn_be:5a:27 (00:01:6c:be:5a:27)
    Type: IP (0x0800)
⊞ Internet Protocol, Src: 172.16.1.50 (172.16.1.50), Dst: tessie.mitht.ru (193.232.216.7)
⊞ Transmission Control Protocol, Src Port: 1365 (1365), Dst Port: 3128 (3128), Seq: 1, Ack: 1, Len: 331
⊞ Hypertext Transfer Protocol
```

Поле	Описание
Destination	**Destination: it-server.class.mitht.ru (00:04:23:bf:bc:19)** Интерпретация аналогична интерпретации информации из заголовка списка.
Source	**Source: Foxconn_be:5a:27 (00:01:6c:be:5a:27)** Интерпретация аналогична интерпретации информации из заголовка списка.
Type	**Type: IP (0x0800)** – На сетевом уровне используется протокол IPv4. Значение, этого поля позволяет устройству определить, какому протоколу сетевого уровня следует дальше передать полученное PDU. В данном случае – это протокол IP. Другие наиболее часто встречающиеся значения поля Type: **0x0806 –** ARP, **0x86DD** – IPv6.

Рисунок 16 – Поля заголовка кадра протокола Ethernet.

Byte 1	Byte 2	Byte 3	Byte 4
Version	**Header length**	**Differentiated Services Field**	**Total Length**
Identification		**Flag**	**Fragment Offset**
Time to Live	**Protocol**	**Header Checksum**	
Source			
Destination			
Options			Padding

```
⊞ Frame 1417 (385 bytes on wire, 385 bytes captured)
⊞ Ethernet II, Src: Foxconn_be:5a:27 (00:01:6c:be:5a:27), Dst: it-server.class.mitht.ru (00:04:23:bf:bc:19)
⊟ Internet Protocol, Src: 172.16.1.50 (172.16.1.50), Dst: tessie.mitht.ru (193.232.216.7)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 371
    Identification: 0xd63d (54845)
  ⊞ Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (0x06)
  ⊞ Header checksum: 0xdc14 [correct]
    Source: 172.16.1.50 (172.16.1.50)
    Destination: tessie.mitht.ru (193.232.216.7)
⊞ Transmission Control Protocol, Src Port: 1365 (1365), Dst Port: 3128 (3128), Seq: 1, Ack: 1, Len: 331
⊞ Hypertext Transfer Protocol
```

Рисунок 17 – Поля заголовка пакета протокола IP

Информацию в заголовке списка можно интерпретировать следующим образом:

Internet Protocol – это пакет протокола IP;

src: 172.16.1.50 (172.16.1.50) – сетевой адрес устройства отправителя 172.16.1.50;

Dst: tessie.mitht.ru (193.232.216.7) – Сетевой адрес устройства получателя;

193.232.216.7 – DNS имя устройства получателя tessie.mitht.ru.

Интерпретация значений наиболее важных полей приведена в таблице 6.

Таблица 6 – Интерпретация значений наиболее важных полей

Поле	Описание
Time to Live	**Time to live: 128** – Максимально возможное количество сетевых устройств, которые могут обработать и передать пакет дальше по сети равняется 128.
Protocol	**Protocol: TCP (0x06)** – На транспортном уровне используется протокол TCP. Значение, этого поля позволяет устройству определить, какому протоколу транспортного уровня следует дальше передать полученное PDU. В данном случае – это протокол TCP. Другие наиболее часто встречающиеся значения поля Protocol: **0x01** – ICMP, **0x11** - UDP
Source	**Source: 172.16.1.50 (172.16.1.50),** Интерпретация аналогична интерпретации информации из заголовка списка.
Destination	**Destination: tessie.mitht.ru (193.232.216.7)** Интерпретация аналогична интерпретации информации из заголовка списка.

1.7 Протокол TCP

Протокол TCP – протокол транспортного уровня, обеспечивающий надёжную передачу информации между приложениями взаимодействующих устройств.

Схематичное изображение заголовка пакета протокола IP и соответствующий вывод программы Wireshark показаны на рисунке 18.

Зелёным цветом выделены поля, выводимые программой.

2 Bytes			2 Bytes
Source port			**Destination Port**
Sequence number			
Acknowledgement number			
Header Length	(Reserved)	**Flags**	**Window Size**
TCP Checksum			Urgent Pointer
Options (if any)			

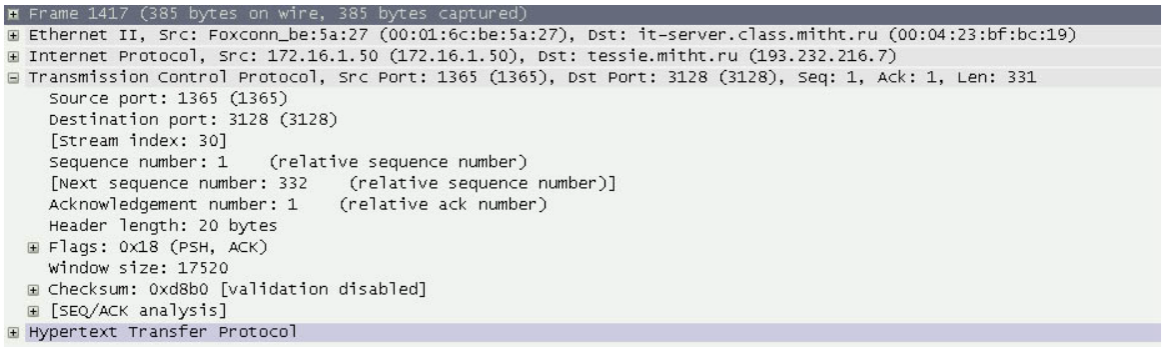


Рисунок 18 - Поля заголовка сегмента TCP

Информацию в заголовке списка можно интерпретировать следующим образом:

Transmission control Protocol, - Это сегмент протокола TCP.

Src Port: 1365 (1365), - Приложение устройства отправителя использует порт 1365.

Dst Port: 3128 (3128), - Приложение устройства получателя использует порт 3128

Len: 331 – Сегмент содержит 331 байт информации.

Интерпретация значений наиболее важных полей приведена в таблице 7.

Таблица 7 – Интерпретация значений наиболее важных полей

Поле	Описание
Source port	**Source Port: 1365 (1365)** Интерпретация аналогична интерпретации информации из заголовка списка.
Destination port	**Destination Port: 3128 (3128)** Интерпретация аналогична интерпретации информации из заголовка списка.
Sequence number и **Acknowledgement number**	**Sequence number: 1 (.relative sequence number)** **[Next sequence number: 332 (relative sequence number)]** **Acknowledgement number: 1 (relative ack number)** Поля, использующиеся для организации надёжной доставки информации между приложениями.
Window size	Количество байт, которые могут быть переданы без подтверждения.

1.8 Протокол UDP

Протокол TCP – протокол транспортного уровня, обеспечивающий передачу информации между приложениями взаимодействующих устройств с минимальным задержками.

Схематичное изображение заголовка пакета протокола IP и соответствующий вывод программы Wireshark показаны на рисунке 19.

Зелёным цветом выделены поля, выводимые программой.

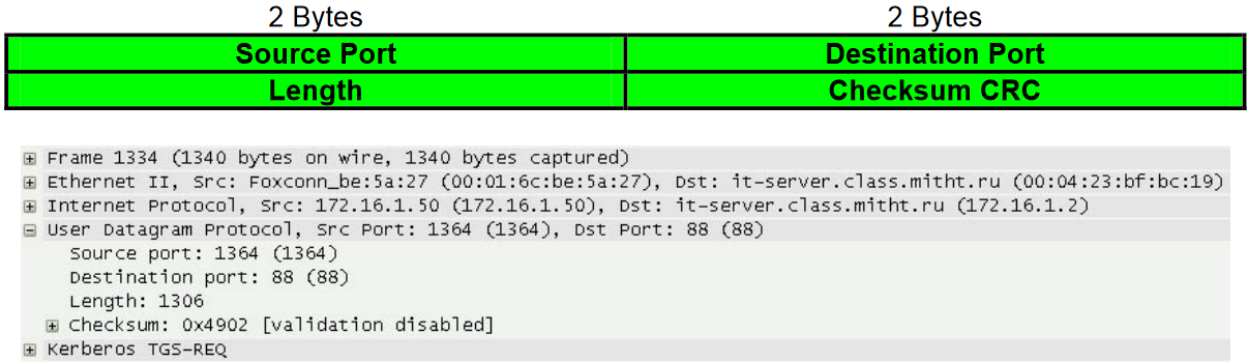


Рисунок 19 – Поля заголовка датаграммы UDP

Информацию в заголовке списка можно интерпретировать следующим образом (таблица 8):

User Datagram Protocol – это датаграмма протокола TCP

Src Port: 1364 (1364) – приложение устройства отправителя использует порт 1364.

Dst Port: 88 (88) – приложение устройства получателя использует порт 88

Таблица 8 – Интерпретация информации в заголовке списка

Поле	Описание
Source port	**Source Port: 1364 (1364)** Интерпретация аналогична интерпретации информации из заголовка списка.
Destination port	**Destination Port: 88 (88)** Интерпретация аналогична интерпретации информации из заголовка списка.
Length	Длина датаграммы.

1.9 Статистическая обработка сетевого трафика

Wireshark позволяет выполнять различную статистическую обработку полученных данных. Все доступные операции находятся в меню «Statistics».

Общая статистика – количество полученных/переданных пакетов, средняя скорость передачи и т.д. доступны через пункт «Statistics->Summary».

Получить информацию по статистике обработанных протоколов в полученных пакетах можно через пункт «Statistics->Protocol Hierarchy».

Статистику по типу ip-пакетов, их размеру и порту назначения можно получить выбрав подпункты меню «IP-address…», «Packet length» и «Port type» соответственно.

Одной из наиболее интересных возможностей является генерация диаграммы взаимодействия между узлами, которая доступна пунктом меню «Flow Graph…». В результате можно наблюдать в достаточно наглядной форме процесс взаимодействия на уровне протоколов. Например, на рисунке 20 приведена диаграмма взаимодействия при получении узлом win2003 статической web-страницы с сервера http://linux.

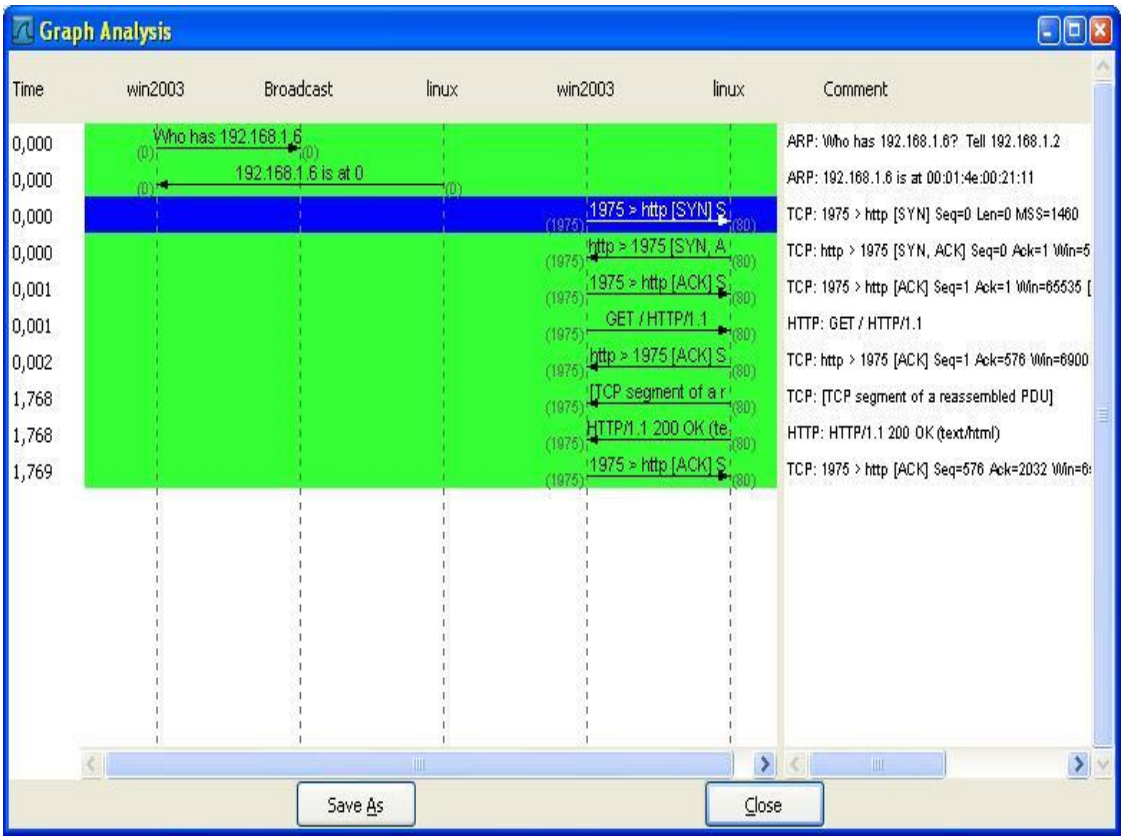


Рисунок 20 – Диаграмма взаимодействия

1.10 Пример исследования протокола с использованием Wireshark

В качестве примера исследования некоторого протокола с использованием сниффера рассмотрим протокол ARP.

Протокол ARP. ARP (англ. Address Resolution Protocol — протокол разрешения адресов) — сетевой протокол, предназначенный для преобразования

IP-адресов (адресов сетевого уровня) в MAC-адреса (адреса канального уровня) в сетях TCP/IP. Он определён в *RFC 826*.

Данный протокол очень распространенный и чрезвычайно важный. Каждый узел сети имеет как минимум два адреса, физический адрес и логический адрес. В сети Ethernet для идентификации источника и получателя информации используются оба адреса. Информация, пересылаемая от одного компьютера другому по сети, содержит в себе физический адрес отправителя, IP-адрес отправителя, физический адрес получателя и IP-адрес получателя. ARP-протокол обеспечивает связь между этими двумя адресами. Существует четыре типа ARP-сообщений: ARP-запрос (ARP request), ARP-ответ (ARP reply), RARP-запрос (RARP-request) и RARP-ответ (RARP-reply). Локальный хост при помощи ARP-запроса запрашивает физический адрес хоста-получателя. Ответ (физический адрес хоста-получателя) приходит в виде ARP-ответа. Хост-получатель, вместе с ответом, шлет также RARP-запрос, адресованный отправителю, для того, чтобы проверить его IP адрес. После проверки IP адреса отправителя, начинается передача пакетов данных.

Перед тем, как создать подключение к какому-либо устройству в сети, IP-протокол проверяет свой ARP-кэш, чтобы выяснить, не зарегистрирована ли в нём уже нужная для подключения информация о хосте-получателе. Если такой записи в ARP-кэше нет, то выполняется широковещательный ARP-запрос. Этот запрос для устройств в сети имеет следующий смысл: «Кто-нибудь знает физический адрес устройства, обладающего следующим IP-адресом?» Когда получатель примет этот пакет, то должен будет ответить: «Да, это мой IP-адрес. Мой физический адрес следующий: ...» После этого отправитель обновит свой ARP-кэш, и будет способен передать информацию получателю.

RARP (англ. Reverse Address Resolution Protocol – обратный протокол преобразования адресов) – выполняет обратное отображение адресов, то есть преобразует аппаратный адрес в IP-адрес.

Протокол применяется во время загрузки узла (например, рабочей станции), когда он посылает групповое сообщение-запрос со своим физическим адресом. Сервер принимает это сообщение и просматривает свои таблицы (либо перенаправляет запрос куда-либо ещё) в поисках соответствующего физическому IP-адреса. После обнаружения найденный адрес отсылается обратно на запросивший его узел. Другие станции также могут «слышать» этот диалог и локально сохранить эту информацию в своих ARP-таблицах.

RARP позволяет разделять IP-адреса между не часто используемыми хост-узлами. После использования каким-либо узлом IP-адреса он может быть освобождён и выдан другому узлу. RARP является дополнением к ARP, и описан в RFC 903.

Для просмотра ARP-кэша можно использовать одноименную утилиту arp с параметром «-a». Например:

```
D:\>arp -a
Interface: 192.168.1.2 --- 0x10003
  Internet Address    Physical Address      Type
```

```
  192.168.1.1          00-15-e9-b6-67-4f     dynamic
  192.168.1.6          00-01-4e-00-21-11     dynamic
```

Из данного результата команды arp видно, что в кэше на данный момент находится 2 записи и видны соответственно ip-адреса машин и MAC-адреса их сетевых адаптеров.

Записи в ARP-кэше могут быть статическими и динамическими. Пример, данный выше, описывает динамическую запись кэша. Хост-отправитель автоматически послал запрос получателю, не уведомляя при этом пользователя. Записи в ARP-кэш можно добавлять вручную, создавая статические (static) записи кэша. Это можно сделать при помощи команды:

```
arp -s <IP адрес> <MAC адрес>
```

Также можно удалять записи из ARP-кэша. Это осуществляется путем следующего вызова:

```
arp -d <IP адрес>
```

После того, как IP-адрес прошёл процедуру разрешения адреса, он остается в кэше в течение 2-х минут. Если в течение этих двух минут произошла повторная передача данных по этому адресу, то время хранения записи в кэше продлевается ещё на 2 минуты. Эта процедура может повторяться до тех пор, пока запись в кэше просуществует до 10 минут. После этого запись будет удалена из кэша и будет отправлен повторный ARP-запрос.

ARP изначально был разработан не только для IP протокола, но в настоящее время в основном используется для сопоставления IP- и MAC-адресов.

Посмотрим же на практике как работает протокол ARP/RARP. Для этого воспользуемся Wireshark для захвата сетевого трафика.

Рассмотрим пример работы протокола ARP при обращении к машине с адресом 192.168.1.5, выполнив запрос с машины с адресом 192.168.1.2. Для успешного эксперимента предварительно очистим arp-кэш командой

```
arp -d 192.168.1.5
```

Для фильтрации ARP/RARP трафика воспользуемся фильтром захвата. В нашем случае это будет простой фильтр

```
arp or rarp
```

Далее запустим захват трафика командой «Start» и выполним обращение к заданной машине, например, «пропинговав» ее:

```
D:\>ping 192.168.1.5
Pinging 192.168.1.5 with 32 bytes of data:
Reply from 192.168.1.5: bytes=32 time<1ms TTL=64
Reply from 192.168.1.5: bytes=32 time<1ms TTL=64
Reply from 192.168.1.5: bytes=32 time<1ms TTL=64
Reply from 192.168.1.5: bytes=32 time<1ms TTL=64
Ping statistics for 192.168.1.5:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Так как на момент начала работы утилиты ping в arp-кэше не было информации о MAC-адресе соответствующего узла, то первоначально система

должна выполнить определение это самого MAC-адреса, сгенерировав ARP-запрос и отослав его в сеть широковещательным пакетом. После чего она будет ожидать ответа от заданного узла.

Посмотрим же, что мы получим на практике. После остановки Wireshark мы должны увидеть результат схожий с тем, что отображен на рисунке 21. В нашем случае мы видим 2 захваченных пакета: ARP-запрос и ARP-ответ.

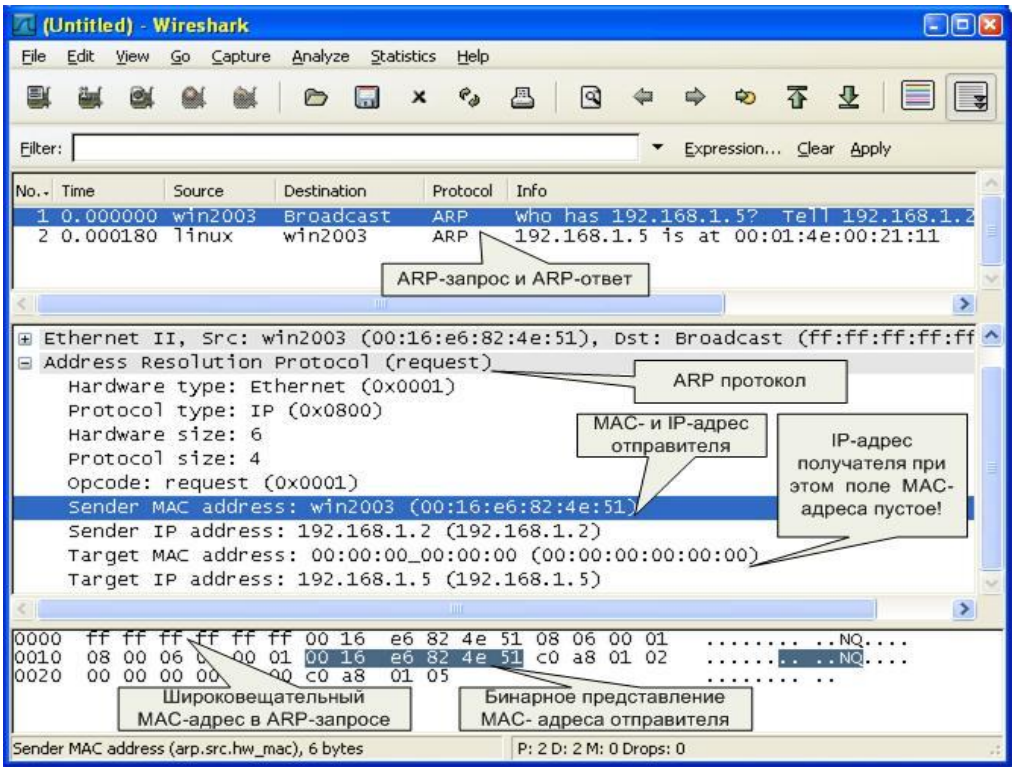


Рисунок 21 – Анализ ARP-запроса

Проанализируем полученные пакеты. Сначала рассмотрим ARP-запрос (пакет №1). Выделив пакет курсором, мы получаем его раскладку по протоколам (Ethernet+ARP) в среднем окне. Wireshark очень наглядно «раскладывает» заголовок протокола по полям.

Мы можем видеть, что в пакете указаны MAC- и IP-адреса отправителя («Sender MAC address» и «Sender IP address» соответственно). Это параметры машины, с которой выполняется запрос. В данном случае запрос направлен на получения («Opcode: request» – запрос) MAC-адреса машины, у которой IP-адрес («Protocol type: IP») 192.168.1.5 («Target IP address»). При этом поле «Target MAC address» обнулено. Так как получатель ARP-запроса на момент запроса не известен, Ethernet-пакет отправляется всем машинам в данном локальном сегменте, о чем сигнализирует MAC адрес Ethernet-пакета «ff:ff:ff:ff:ff:ff».

Примечание. Обратите внимание, что пакет представляет собой бинарную последовательность и Wireshark выполняет большую работу по преобразованию полей из бинарного представления в удобочитаемый вариант.

Все работающие машины в сети получают пакет с ARP-запросом, анализируют его, а ответ отсылает только та машина, чей IP-адрес соответствует IP-адресу в запросе. Таким образом, второй полученный пакет является ARP-ответом (см. рисунок 22). Это следует из параметра поля «Opcode: reply». Обратите внимание, что данный пакет был отправлен именно той машиной, чей MAC-адрес нас и интересовал («Sender IP address: 192.168.1.5»). При этом поле «Sender MAC address» заполнено значением «00:01:4E:00:21:11».

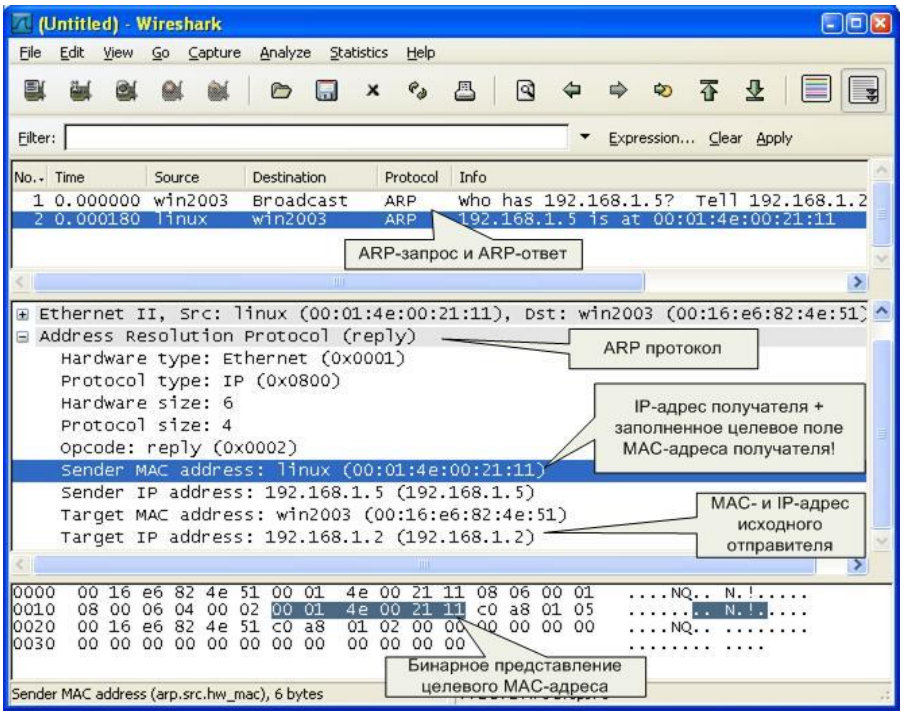


Рисунок 22 – Анализ ARP-ответа

Примечание. Обратите внимание на поле «Info» в списке захваченных пакетов. Wireshark и тут упрощает анализ сетевого трафика, подсказывая назначение пакетов.

Теперь мы можем повторно просмотреть ARP-кэш и сверить данные в нем с данными, которые мы узнали из анализа пакетов ARP-запрос/ответа:

```
D:\>arp -a
Interface: 192.168.1.2 --- 0x10003
  Internet Address      Physical Address      Type
  192.168.1.5           00-01-4e-00-21-11     dynamic
```

Стоит также отметить, что в реальных условиях в локальной сети с большим количеством машин arp/rarp трафик бывает гораздо более интенсивным.

2 Этапы старта программы

Запустить программу Wireshark. Для запуска программы необходимо нажать: Пуск > Программы > Wireshark, либо два раза щёлкнуть левой кнопкой мыши по ярлыку программы на рабочем столе.

Настроить параметры захвата сетевого трафика. Для настройки параметров захвата сетевого трафика необходимо:

1) Щелчком левой кнопки мыши по кнопке Capture Options вызвать меню настроек (рисунок 20).

Рисунок 20 – Кнопка Capture Options

2) Установить параметры в соответствии с рисунком 23.

Следующие опции должны быть активированы:

- Capture packets in promiscuous mode Update list of packets in real time;
- Automatic scrolling in live capture;
- Enable MAC name resolution;
- Enable network name resolution.

В качестве интерфейса, используемого для захвата трафика выбрать физический (не виртуальный) адаптер и установить тип адаптера Local.

Запустить процесс захвата трафика. Для запуска процесса необходимо нажать кнопку Start в меню настроек.

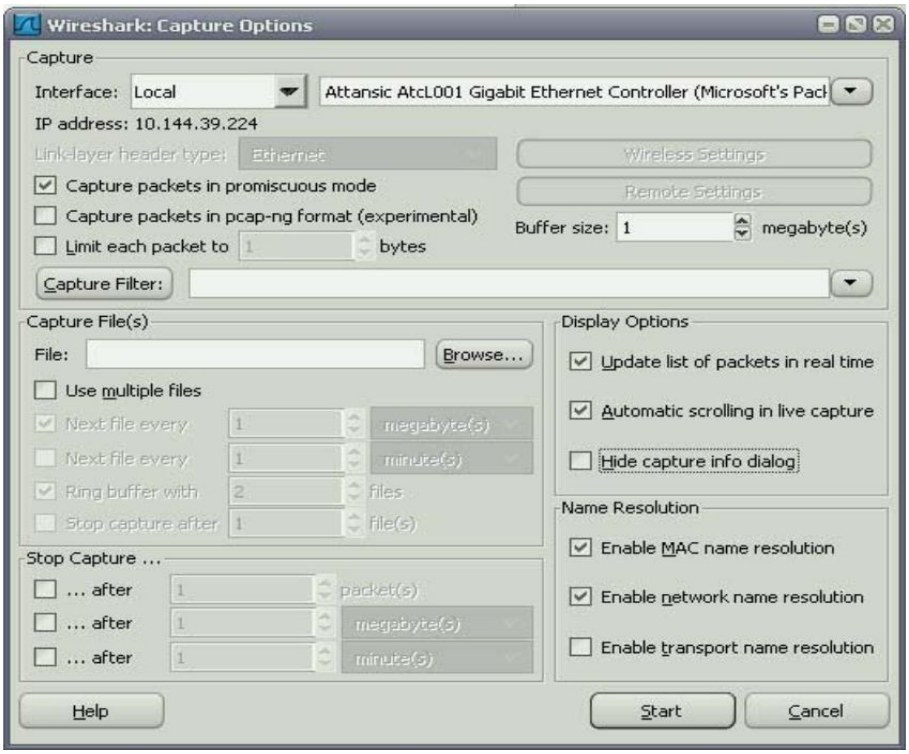


Рисунок 23 – Окно настроек

3 Лабораторная работа №1. Анализ сетевого трафика и протоколов (на базе WireShark)

Цель работы: приобретение навыков анализа сетевого трафика компьютерных сетей; изучение структуры сетевых протоколов различных уровней

Задание на выполнение

1. Изучить краткие теоретические сведения по возможностям, приемам работы с программой Wireshark.
2. Изучить: типы фильтрации трафика, правила построения фильтров, приемы статистической обработки сетевого трафика в Wireshark.
3. Получить вариант задания (адрес url, таблица 9) у преподавателя. Запустив Wireshark на захват, выполнить загрузку страницы согласно варианту, приведенному ниже
4. Остановить и сохранить захват. Для захваченных пакетов определить статистические данные:
- процентное соотношение трафика разных протоколов в сети;
- среднюю скорость кадров/сек;
- среднюю скорость байт/сек;
- минимальный, максимальный и средний размеры пакета;
- степень использования полосы пропускания канала (загрузку сети).
5. Отфильтровать в захвате IP пакеты. Определить статистические данные:
- процентное соотношение трафика разных протоколов стека tcp/ip в сети;
- средний, минимальный, максимальный размеры пакета.
6. На примере третьего захваченного IP-пакета указать структуры протокола канального уровня (протокола Ethernet 802.3, Wi-Fi 802.11, либо другого, используемого в вашей конфигурации) и протокола IPv4. Отметить поля заголовков и описать их и интерпретировать их значения.
7. Запустив Wireshark на захват, выполнить команду ping для IP адреса компьютера (предварительно определив его адрес с помощью ipconfig; пример команды: ping 172.17.20.246). Сохранить результат. Сформировав нужный фильтр, отфильтровать пакеты, относящиеся к выполнению команды ping. На базе полученных пакетов и значений их полей интерпретировать результат работы утилиты ping. Описать все протоколы, используемые утилитой.

Таблица 9 – Возможные варианты заданий

Вариант	Адрес url
1	https://www.google.ru
2	https://lenta.ru
3	https://sputnik.by
4	https://www.belta.by
5	https://www.microsoft.com
6	https://ria.ru
7	https://www.lg.com
8	https://hyundai.by
9	https://www.canon.ru
10	https://av.by
11	https://www.samsung.com
12	https://www.huawei.com
13	https://auto.ru
14	https://www.audi.com
15	https://www.bmw.com
16	https://skoda.com

8. Сформировать не менее 3-х сложных фильтров захвата с использованием полей протоколов, операторов сравнения (таблицы 1 и 2 из файла теоретические_указания4) и логических операторов; каждый раз перезапуская захват, для каждого фильтра захватить соответствующие пакеты.

9. Выполнить анализ ARP-протокола по примеру из методических указаний.

10. Выполнить анализ TCP-сеансов по примеру из методических указаний

11. Оформить отчет, содержащий титульный лист, данные варианта задания, скриншоты полученных результатов.

Список использованных источников

1. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для ВУЗов. Юбилейное издание. – СПб.: Питер, 2020. – 1008 с.

2. Уколов С.С., Таваева А.Ф. Сети и системы телекоммуникаций. – Екатеринбург, 2018. – 63 с.

3. Таненбаум, Э. Компьютерные сети / Э. Таненбаум; Пер. с англ. А. Леонтьева. – 3-е изд. – М.; СПб.; Н. Новгород и др. : Питер, 2002. – 846 с.

2.4 ЛАБОРАТОРНЫЙ ПРАКТИКУМ. ЧАСТЬ 4. ПОСТРОЕНИЕ И КОНФИГУРИРОВАНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ

ОГЛАВЛЕНИЕ

1 Краткие теоретические сведения

Открытая система(OSI) — это стандартизированный набор протоколов и спецификаций, который гарантирует возможность взаимодействия оборудования различных производителей. Она реализуется набором модулей, каждый из которых решает простую задачу внутри элемента сети. Каждый из модулей связан с одним или несколькими другими модулями. Решение сложной задачи подразумевает определенный порядок следования решения простых задач, при котором образуется многоуровневая иерархическая структура на рисунке 1. Это позволяет любым двум различным системам связываться независимо от их основной архитектуры.

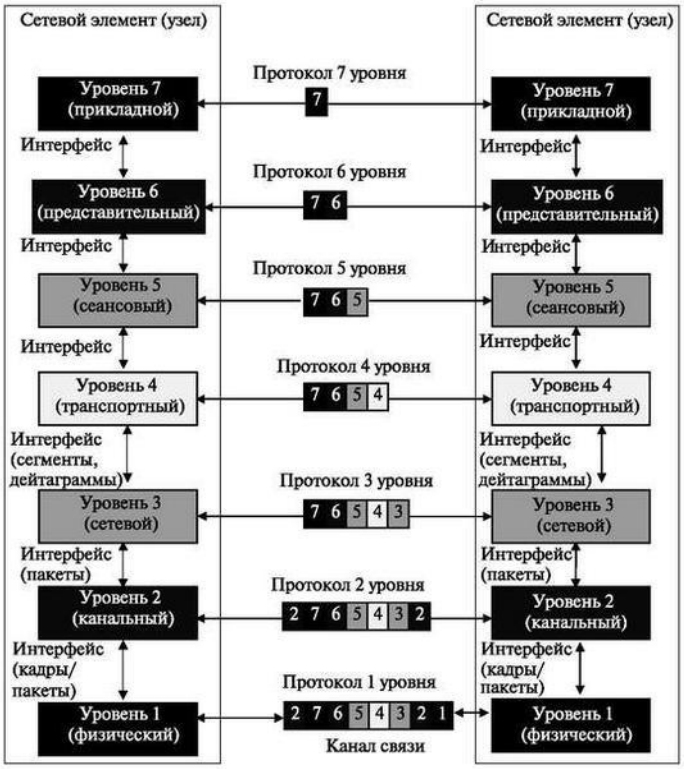


Рисунок 1 – Модель взаимодействия открытых систем OSI

Обмен информацией между модулями происходит на основе определенных соглашений, которые называются интерфейсом. При передаче сообщения модуль верхнего уровня решает свою часть задачи, а результат, понятный только ему, оформляет в виде дополнительного поля к исходному сообщению (заголовка) и передает измененное сообщение на дообслуживание в нижележащий уровень. Этот процесс называется инкапсуляцией.

1.1 Стек протоколов компьютерной сети Интернет. IP-адресация

Стек протоколов сети Интернет был разработан до модели OSI. Поэтому уровни в стеке протоколов Интернета не соответствуют аналогичным уровням в модели OSI. Стек протоколов Интернета состоит из пяти уровней: физического, звена передачи данных, сети, транспортного и прикладного. Первые четыре уровня обеспечивают физические стандарты, сетевой интерфейс, межсетевое взаимодействие и транспортные функции, которые соответствуют первым четырем уровням модели OSI. Три самых верхних уровня в модели OSI представлены в стеке протоколов Интернета единственным уровнем, называемым прикладным уровнем (рисунок 2).

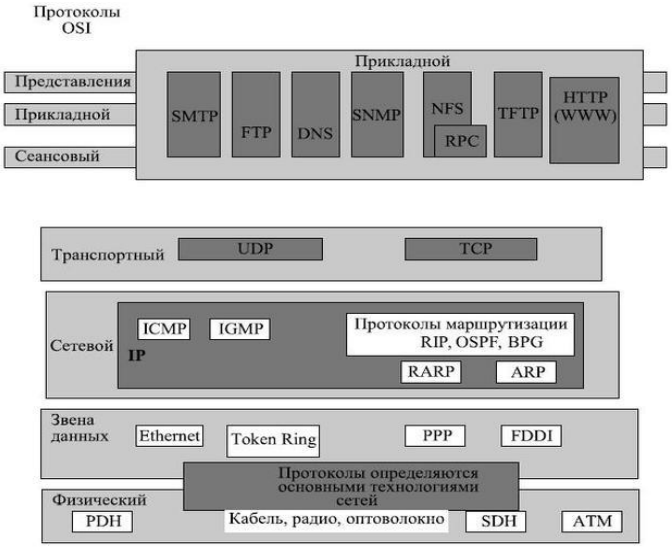


Рисунок 2 – Стек протоколов Интернета по сравнению с OSI

Стек базовых протоколов Интернета — иерархический, составленный из диалоговых модулей, каждый из которых обеспечивает заданные функциональные возможности; но эти модули не обязательно взаимозависимые. В отличие от модели OSI, где определяется строго, какие функции принадлежат каждому из ее уровней, уровни набора протокола TCP/IP содержат относительно независимые протоколы, которые могут быть смешаны и согласованы в зависимости от потребностей системы. Термин иерархический означает, что каждый верхний протокол уровня поддерживается соответственно одним или более протоколами нижнего уровня.

На транспортном уровне стек определяет два протокола: протокол управления передачей (TCP) и протокол пользовательских дейтаграмм (UDP). На сетевом уровне — главный протокол межсетевого взаимодействия (IP), хотя на этом уровне используются некоторые другие протоколы, о которых будет сказано ниже.

Адресация узлов в IP-сетях. В сетях TCP/IP принято различать адреса сетевых узлов трех уровней:

- физический (или локальный) адрес узла (MAC-адрес сетевого адаптера или порта маршрутизатора); эти адреса назначаются производителями сетевого оборудования;

- IP-адрес узла (например, 192.168.0.1), данные адреса назначаются сетевыми администраторами или Интернет-провайдерами;

- символьное имя (например, www.microsoft.com); эти имена также назначаются сетевыми администраторами компаний или Интернет-провайдерами.

Рассмотрим подробнее IP-адресацию (рисунок 3). Компьютеры или другие сложные сетевые устройства, подсоединенные к нескольким физическим сетям, имеют несколько IP-адресов — по одному на каждый сетевой интерфейс. Схема адресации позволяет проводить единичную, широковещательную и групповую адресацию.

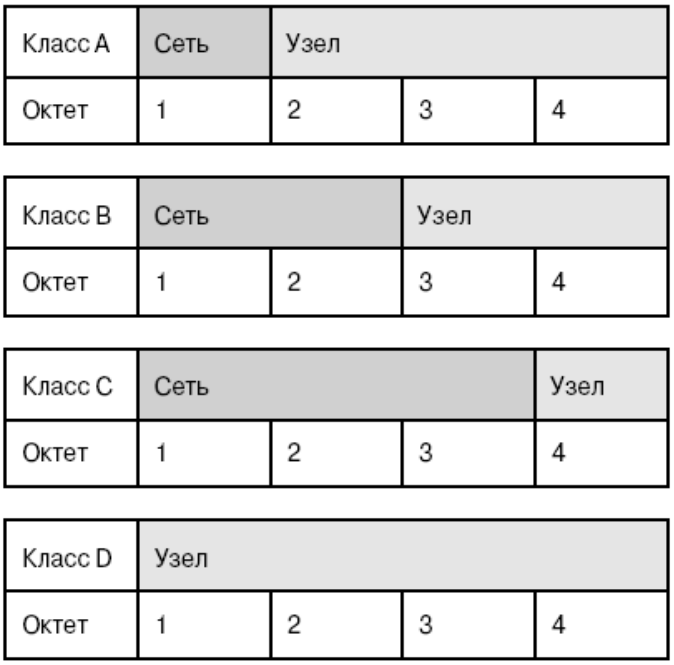


Рисунок 3 – Классы IP-сетей

Таким образом, выделяют 3 типа IP-адресов:

- Unicast-адрес (единичная адресация конкретному узлу) — используется в коммуникациях "один-к-одному";

- Broadcast-адрес (широковещательный адрес, относящийся ко всем адресам подсети) — используется в коммуникациях "один-ко-всем". В этих адресах поле идентификатора устройства заполнено единицами. IP-адресация допускает широковещательную передачу, но не гарантирует ее — эта возможность зависит от конкретной физической сети. Например, в сетях Ethernet широковещательная передача выполняется с той же эффективностью, что и обычная передача данных, но есть сети, которые вообще не поддерживают такой тип передачи или поддерживают весьма ограничено;

- Multicast-адрес (групповой адрес для многоадресной отправки пакетов) — используется в коммуникациях "один-ко-многим". Поддержка групповой

адресации используется во многих приложениях, например, приложениях интерактивных конференций. Для групповой передачи рабочие станции и маршрутизаторы используют протокол IGMP, который предоставляет информацию о принадлежности устройств определенным группам. В таблице 1 представлены диапазоны IP-адресов для различных классов IP-сетей.

Таблица 1 – Диапазоны IP-адресов для различных классов IP-сетей.

Класс сети	Старшие биты адреса	Наименьший идентификатор сети	Наибольший идентификатор сети	Количество сетей
Класс A	0	1.0.0.0	126.0.0.0	126
Класс B	10	128.0.0.0	191.255.0.0	16 384
Класс C	11	192.0.0.0	223.255.255.0	2 097 152

1.2 Назначение масок подсетей

Чтобы выделить подсеть, биты сетевого узла должны быть переназначены как сетевые биты посредством деления октета (или октетов) сетевого узла на части. Такой механизм часто называют заимствованием битов, но более точным термином будет аренда битов, хотя последний используется очень редко. Процесс деления всегда начинается с крайнего левого бита узла, положение которого зависит от класса IP-адреса.

Помимо повышения управляемости, создание подсетей позволяет сетевым администраторам ограничить широковещательные рассылки и реализовать механизм низкоуровневой безопасности в локальной сети. Безопасность при использовании подсетей в локальных сетях реализуется благодаря тому, что доступ в другие подсети организуется через маршрутизаторы. Маршрутизатор может быть настроен таким образом, чтобы разрешить или запретить доступ к подсети на основе различных критериев, реализуя таким образом политику безопасности.

Чтобы определить требуемое количество битов для создания подсети, разработчик сети должен рассчитать, какое максимальное число узлов будет в подсети, и общее количество подсетей. Независимо от класса IP-адреса, последние 2 бита в последнем октете никогда не могут быть использованы для формирования подсети. Они называются наименее значимыми битами. Заимствование всех доступных битов, за исключением двух последних, позволяет создать подсеть, которая содержит только два узла. Чтобы создать маску подсети, дающую маршрутизатору информацию, необходимую для вычисления адреса подсети, которой принадлежит конкретный узел, необходимо выбрать столбец из таблицы с нужным количеством битов и в качестве значения маски воспользоваться числом строкой выше из того же столбца, как показано в табл. ниже. Это значение получено в результате сложения двоичных значений для знакомест используемых

битов. Как показано на рисунке 4, если заимствованы 3 бита, маска подсети для сети класса С будет равна 255.255.255.224. При использовании формата записи маски с обратной косой чертой он может быть представлен как ‘‘/27’’. Число, указанное после символа обратной косой черты, представляет собой количество битов, составляющих адрес сети, плюс биты, использующиеся для маски подсети.

Формат с обратной косой чертой	/25	/26	/27	/28	/29	/30	—	—
Маска	128	192	224	240	248	252	254	255
Бит	1	2	3	4	5	6	7	8
Значение	128	64	32	16	8	4	2	1

Рисунок 4 – IP маски

Еще одним способом вычислить маску подсети и количество доступных подсетей

и узлов является использование формул, которые приведены и объяснены ниже.

Количество доступных подсетей равно 2 в степени, равной количеству используемых для формирования подсети битов, минус 2:

($2^{\text{количество заимствованных битов}}$) - 2 = количество используемых подсетей.

В качестве примера предположим, что необходимо разместить по 30 узлов в 5-ти подсетях.

При заимствовании трех битов из узловой части сети класса С количество используемых подсетей 2^3-2=6.

Количество доступных узлов равно 2 в степени, равной количеству оставшихся от заимствования битов, минус 2:

($2^{\text{оставшиеся биты}}$) - 2 = количество используемых узлов.

Например, при заимствовании трех битов из узловой части сети класса С для адресации узлов будут использоваться 5 битов, следовательно, количество узлов в каждой подсети равно 2^5 - 2 = 30.

Таким образом, маска - 255.255.255.224 или «/27».

2 Изучение пакета Cisco Packet Tracer

Cisco Packet Tracer — эмулятор сети передачи данных, выпускаемый фирмой Cisco Systems. Позволяет делать работоспособные модели сетей типа Ethernet, Serial, ISDN, Frame Relay, настраивать (командами Cisco IOS) маршрутизаторы и коммутаторы, взаимодействовать между несколькими пользователями (через облако). Включает в себя серии маршрутизаторов Cisco 1800, 2600, 2800 и коммутаторов 2950, 2960, 3650 и др. Кроме того, имеются серверы DHCP, HTTP, TFTP, FTP, рабочие станции, различные модули к компьютерам и маршрутизаторам, устройства WiFi, различные кабели.

Успешно позволяет создавать сложные макеты сетей, проверять на работоспособность топологии.

После запуска Cisco Packet Tracer общий вид программы можно увидеть на риунке 5.

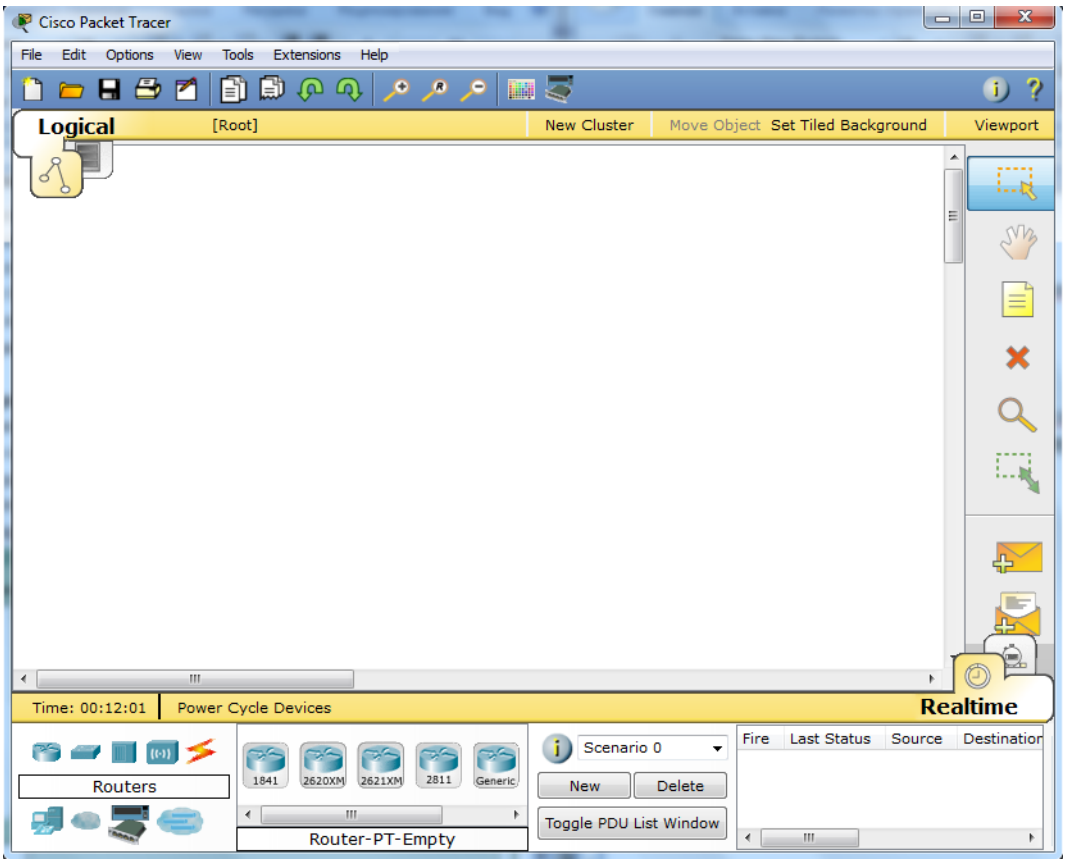


Рисунок 5 – Общий вид программы Packet Tracer.

Рабочая область окна программы состоит из следующих элементов:

1. **Menu Bar** - Панель, которая содержит меню File, Edit, Options, View, Tools, Extensions, Help.

2. **Main Tool Bar** - содержит графические изображения ярлыков для доступа к командам меню File, Edit, View и Tools, а также кнопку Network Information.

3. **Common Tools Bar** - Панель, которая обеспечивает доступ к наиболее используемым инструментам программы: Select, Move Layout, Place Note, Delete, Inspect, Add Simple PDU и Add Complex PDU.

4. **Logical/Physical Workspace and Navigation Bar** - Панель, которая дает возможность переключать рабочую область: фізическую или логическую, а также позволяет перемещаться между уровнями кластера.

5. **Workspace** - Область, в которой происходит создание сети, проводятся наблюдения за симуляцией и просматривается разная информация и статистика.

6. **Realtime/Simulation Bar** - С помощью закладок этой панели можно переключаться между режимом Realtime и режимом Simulation. Она также содержит кнопки, относящиеся к Power Cycle Devices, кнопки Play Control и переключатель Event List в режиме Simulation.

7. **Network Component Box** - Это область, в которой выбираются устройства и связи для размещения их на рабочем пространстве. Она содержит область Device-Type Selection и область Device-Specific Selection.

8. **Device-Type Selection Box** - Эта область содержит доступные типы устройств и связей в Packet Tracer. Область Device-Specific Selection изменяется в зависимости от выбранного устройства

9. **Device-Specific Selection Box** - Эта область используется для выбора конкретных устройств и соединений, необходимых для постройки в рабочем пространстве сети.

10. **User Created Packet Window** - Это окно управляет пакетами, которые были созданы в сети во время симуляции сценария.

Для создания топологии необходимо выбрать устройство из панели Network Component, а затем из панели Device-Type Selection выбрать тип выбранного устройства. После этого нужно нажать левую кнопку мыши в поле рабочей области программы (Workspace). Также можно переместить устройство прямо из области Device-Type Selection, но при этом будет выбрана модель устройства по умолчанию.

Для быстрого создания нескольких экземпляров одного и того же устройства нужно, удерживая кнопку Ctrl, нажать на устройство в области Device-Specific Selection и отпустить кнопку Ctrl. После этого можно несколько раз нажать на рабочей области для добавления копий устройства.

2.1 Типы устройств, представленные в Cisco Packet Tracer

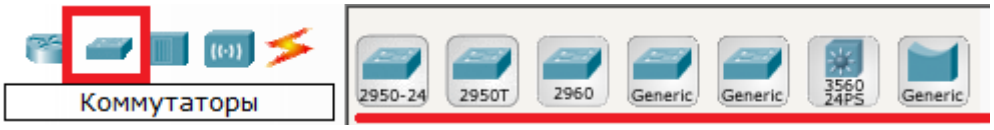
Ниже приведены типы устройств, представленные в пакете Cisco Packet Tracer.

Маршрутизаторы

Маршрутизаторы используется для поиска оптимального маршрута передачи данных на основании специальных алгоритмов маршрутизации, например выбор маршрута (пути) с наименьшим числом транзитных узлов.

Работают на сетевом уровне модели OSI.

Коммутаторы



Коммутаторы - это устройства, работающие на канальном уровне модели OSI и предназначенные для объединения нескольких узлов в пределах одного или нескольких сегментах сети. Передаёт пакеты коммутатор на основании внутренней таблицы - таблицы коммутации, следовательно трафик идёт только на тот MAC-адрес, которому он предназначается, а не повторяется на всех портах (как на концентраторе).

Концентраторы

Концентратор повторяет пакет, принятый на одном порту на всех остальных портах.

Беспроводные устройства

Беспроводные технологии Wi-Fi и сети на их основе. Включает в себя точки доступа.

Физическая комплектация оборудования. Для физической комплектации оборудования необходимыми модулями можно воспользоваться следующим примером.

Установите в рабочем поле маршрутизатор Cisco 1841. В настройках на роутере открываем его **физическую конфигурацию** (рисунок 6).

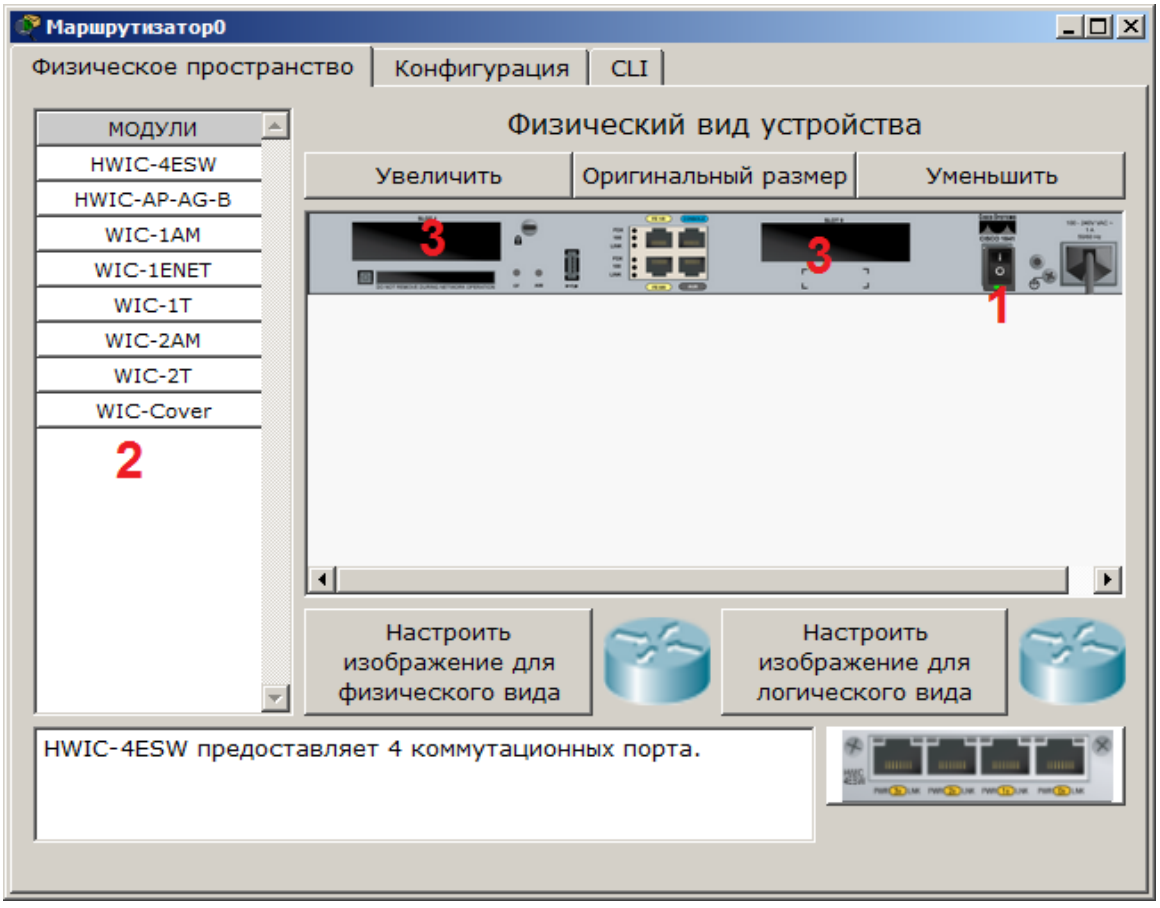


Рисунок 6 – Настройка физической комплектации маршрутизатора

Слева, как мы видим, список модулей (цифра 2), которыми можно укомплектовать данный роутер. Сейчас в нем 2 пустоты (цифра 3). В них можно установить эти модули. Эту операцию нужно производить при выключенном питании (цифра 1).

Модули WIC (HWIC, VWIC) это платы расширения, увеличивающие функционал устройства:

Ниже представлена информация о каждом модуле:

HWIC - 4ESW - высокопроизводительный модуль с 4-мя коммутационными портами Ethernet под разъем RJ-45. Позволяет сочетать в маршрутизаторе возможности коммутатора;

HWIC-AP-AG-B - это высокоскоростная WAN-карта, обеспечивающая функционал встроенной точки доступа для роутеров линейки Cisco 1800 (модульных), Cisco 2800 и Cisco 3800. Данный модуль поддерживает радиоканалы Single Band 802.11b/g или Dual Band 802.11a/b/g;

WIC-1AM включает в себя два разъема RJ-11 (телефоннка), используемых для подключения к базовой телефонной службе. Карта использует один порт для соединения с телефонной линией, другой может быть подключен к аналоговому телефону для звонков во время простоя модема;

WIC-1ENET - это однопортовая 10 Мб/с Ethernet карта для 10BASE-T Ethernet LAN;

WIC-1T предоставляет однопортовое последовательное подключение к удаленным офисам или устаревшим серийным сетевым устройствам, например SDLC концентраторам, системам сигнализации и устройствам packet over SONET (POS);

WIC-2AM содержит два разъема RJ-11, используемых для подключения к базовой телефонной службе. В WIC-2AM два модемных порта, что позволяет использовать оба канала для соединения одновременно.

WIC-2T - 2-портовый синхронный/асинхронный серийный сетевой модуль предоставляет гибкую поддержку многих протоколов с индивидуальной настройкой каждого порта в синхронный или асинхронный режим. Применения для синхронной/асинхронной поддержки представляют:

- низкоскоростную агрегацию (до 128 Кб/с);
- поддержку dial-up модемов;
- синхронные или асинхронные соединения с портами управления другого оборудования и передачу устаревших протоколов типа Bi-sync и SDLC4;

WIC-Cover - стенка для WIC слота, необходима для защиты электронных компонентов и для улучшения циркуляции охлаждающего воздушного потока.

Для изменения комплектации оборудования необходимо:

- отключить питание, кликнув мышью на кнопке питания, перетащить мышью модуль **4ESW** в свободный слот и включить питание. Подождать окончания загрузки роутера. В конфигурации GUI можем увидеть появившиеся 4 новых интерфейса (рисунок 7).

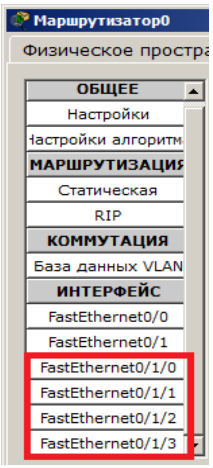


Рисунок 7 – Изменение комплектации оборудования

Остальные устройства комплектуются аналогично. Добавляются новые модули Ethernet (10/100/1000), оптоволоконные разъемы нескольких типов, адаптеры беспроводной сети и др.

Добавим необходимые элементы в рабочую область программы так, как показано на рисунке 8.

При добавлении каждого элемента пользователь имеет возможность дать ему имя и установить необходимые параметры. Для этого необходимо нажать на нужный элемент левой кнопкой мыши (ЛКМ) и в диалоговом окне устройства перейти к вкладке **Config**.

Диалоговое окно свойств каждого элемента имеет две вкладки:

- Physical – содержит графический интерфейс устройства и позволяет симулировать работу с ним на физическом уровне;
- Config – содержит все необходимые параметры для настройки устройства и имеет удобный для этого интерфейс.

Также в зависимости от устройства, свойства могут иметь дополнительную вкладку для управления работой выбранного элемента: Desktop (если выбрано конечное устройство) или CLI (если выбран маршрутизатор) и т.д.

Для удаления ненужных устройств с рабочей области программы используется кнопка Delete (Del).

Добавленные элементы мы свяжем с помощью соединительных связей. Для этого необходимо выбрать вкладку **Connections** из панели Network Component Box. Мы увидим все возможные типы соединений между устройствами. Выберем подходящий тип кабеля. Указание мыши изменится на курсор “connection” (имеет вид разъема). Нажмем на первом устройстве и выбрать соответствующий интерфейс, к которому нужно выполнить соединение, а затем нажмем на второе устройство, выполнив ту же операцию. Можно также соединить с помощью **Automatically Choose Connection Type** (автоматически соединяет элементы в сети). Выберем и нажмем на каждом из устройств, которые нужно соединить. Между устройствами появится кабельное соединение, а индикаторы на каждом конце покажут статус соединения (для интерфейсов которые имеют индикатор).

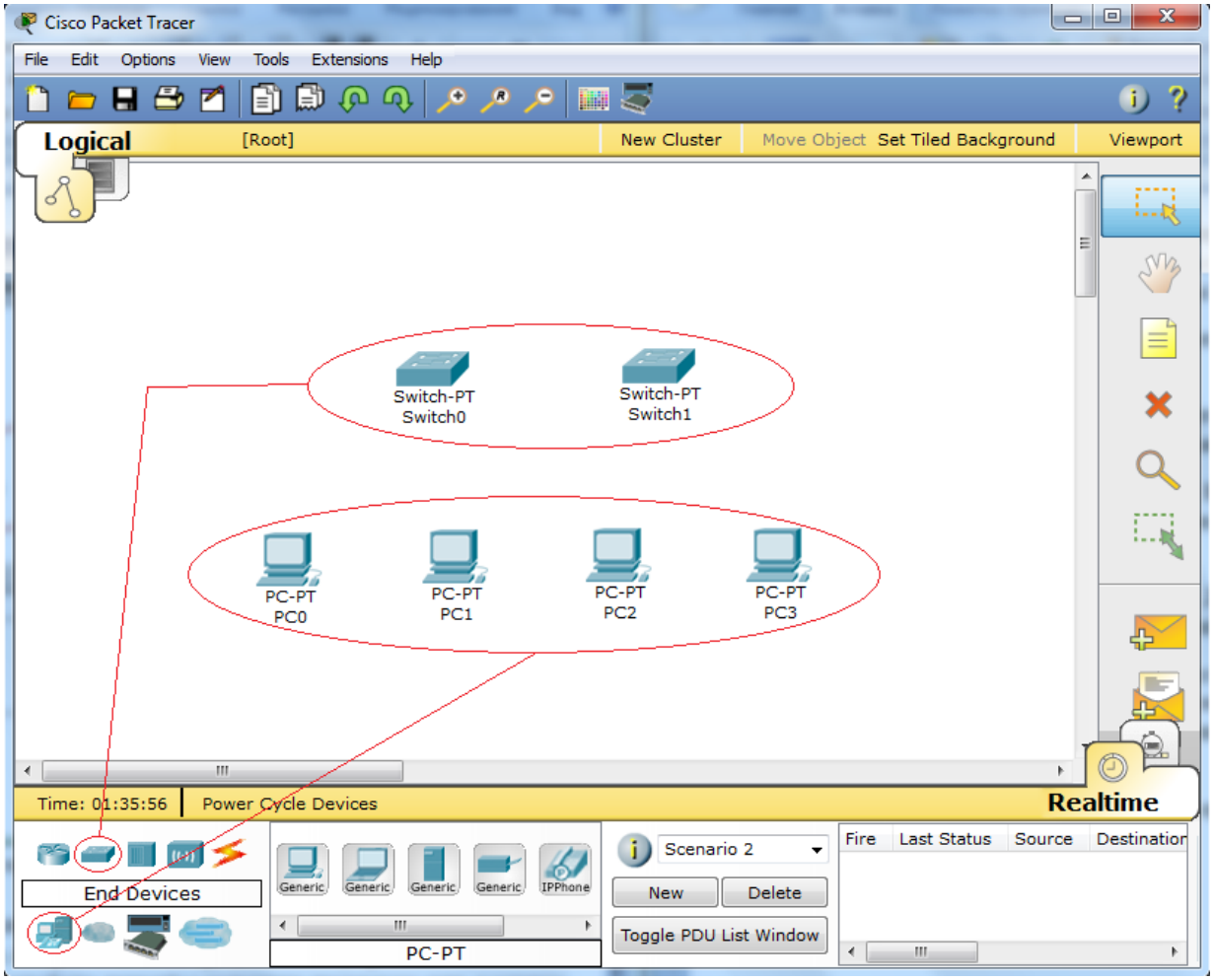


Рисунок 8 – Добавление элементов сети

Packet Tracer поддерживает широкий диапазон сетевых соединений (таблица 2). Каждый тип кабеля может быть соединен лишь с определенными типами интерфейсов.

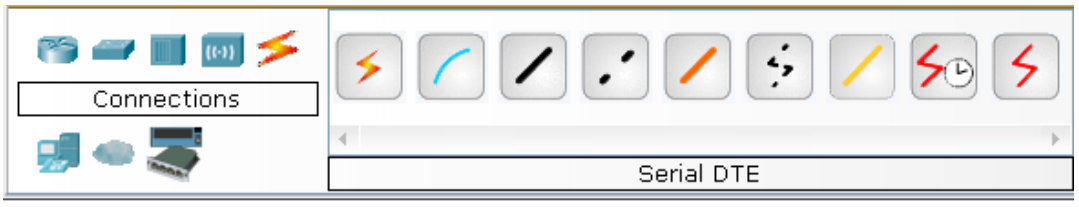


Рисунок 9 – Поддерживаемые в Packet Tracer типы кабелей

Таблица 2 – Типы соединений в Packet Tracer

Тип кабеля	Описание
Console 	Консольное соединение может быть выполнено между ПК и маршрутизаторами или коммутаторами. Должны быть выполнены некоторые требования для работы консольного сеанса с ПК: скорость соединения с обеих сторон должна быть одинаковая, должно быть 7 бит данных (или 8 бит) для обеих сторон, контроль четности должен быть одинаковый, должно быть 1 или 2 стоповых бита (но они не обязательно должны быть одинаковыми), а поток данных может быть чем угодно для обеих сторон.
Copper Straigt-through 	Этот тип кабеля является стандартной средой передачи Ethernet для соединения устройств, который функционирует на разных уровнях OSI. Он должен быть соединен со следующими типами портов: медный 10 Мбит/с (Ethernet), медный 100 Мбит/с (Fast Ethernet) и медный 1000 Мбит/с (Gigabit Ethernet).
Copper Cross-over 	Этот тип кабеля является средой передачи Ethernet для соединения устройств, которые функционируют на одинаковых уровнях OSI. Он может быть соединен со следующими типами портов: медный 10 Мбит/с (Ethernet), медный 100 Мбит/с (Fast Ethernet) и медный 1000 Мбит/с (Gigabit Ethernet).
Fiber 	Оптоволоконная среда используется для соединения между оптическими портами (100 Мбит/с или 1000 Мбит/с).
Тип кабеля	Описание
Phone 	Соединение через телефонную линию может быть осуществлено только между устройствами, имеющими модемные порты. Стандартное представление модемного соединения - это конечное устройство (например, ПК), дозванивающееся в сетевое облако.
Coaxial 	Коаксиальная среда используется для соединения между коаксиальными портами, такие как кабельный модем, соединенный с облаком Packet Tracer.
Serial DCE and DTE  	Соединения через последовательные порты, часто используются для связей WAN. Для настройки таких соединений необходимо установить синхронизацию на стороне DCE-устройства. Синхронизация DTE выполняется по выбору. Сторону DCE можно определить по маленькой иконке “часов” рядом с портом. При выборе типа соединения Serial DCE, первое устройство, к которому применяется соединение, становиться DCE-устройством, а второе - автоматически станет стороной DTE. Возможно и обратное расположение сторон, если выбран тип соединения Serial DTE.

После создания сети ее нужно сохранить, выбрав пункт меню File -> Save или иконку Save на панели **Main Tool Bar**. Файл сохраненной топологии имеет расширение *.pkt .

Packet Tracer дает возможность симулировать работу с интерфейсом командной строки (ИКС) операционной системы IOS, установленной на всех коммутаторах и маршрутизаторах компании Cisco.

Подключившись к устройству, можно работать с ним так, как за консолью реального устройства. Симулятор обеспечивает поддержку практически всех команд, доступных на реальных устройствах.

Подключение к ИКС коммутаторов или маршрутизаторов можно произвести, нажав на необходимое устройство и перейдя в окно свойств к вкладке CLI.

Для симуляции работы командной строки на конечном устройстве (компьютере) необходимо в свойствах выбрать вкладку Desktop, а затем нажать на ярлык Command Prompt.

Работа с файлами в эмуляторе. Packet Tracer дает возможность пользователю хранить конфигурацию некоторых устройств, таких как маршрутизаторы или свичи, в текстовых файлах. Для этого необходимо перейти к свойствам необходимого устройства и во вкладке Config нажать на кнопку “Export…” для экспорта конфигурации Startup Config или Running Config. Так получим диалоговое окно для сохранения необходимой конфигурации в файл, который будет иметь расширение *.txt . Текст файла с конфигурацией устройства running-config.txt (имя по умолчанию) представляется аналогичным к тексту информации полученному при использовании команды show running-config в IOS устройства.

Необходимо отметить что конфигурация каждого устройства сохраняется в отдельном текстовом файле. Пользователь также имеет возможность изменять конфигурацию в сохраненном файле вручную с помощью произвольного текстового редактора. Для предоставления устройству сохраненных или отредактированных настроек нужно во вкладке Config нажать кнопку “Load...” для загрузки необходимой конфигурации Stàrtup Config или кнопку “Merge…” для загрузки конфигурации Running Config.

2.2 Пример создания и настройки конфигурации сети

Добавим на рабочую область программы 2 коммутатора Switch-PT. По умолчанию они имеют имена – Switch0 и Switch1.

Добавим на рабочее поле четыре компьютера с именами по умолчанию PC0, PC1, PC2, PC3.

Соединим устройства в сеть Ethernet , как показано на рисунке 10.

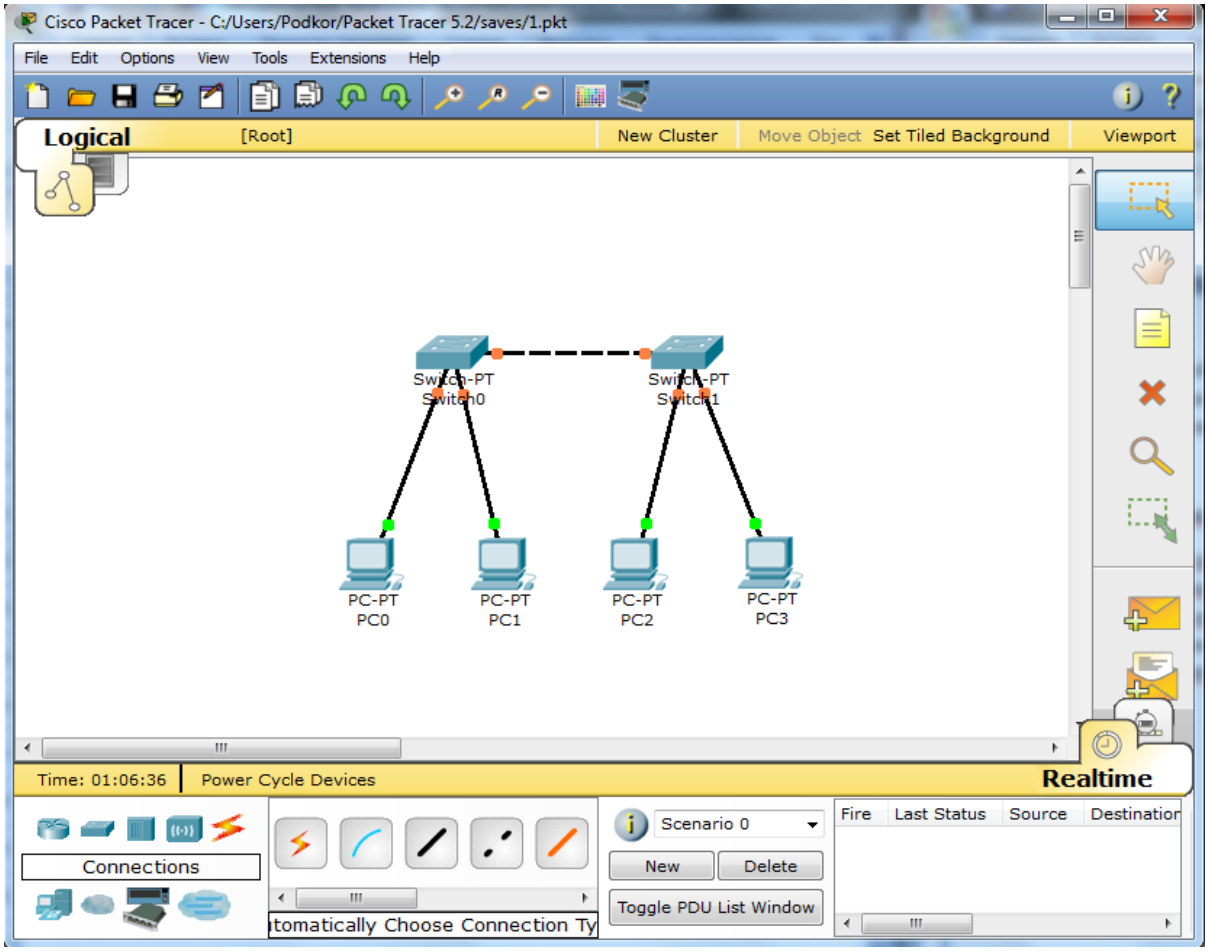


Рисунок 10 – Пример настройки конфигурации сети

Сохраним созданную топологию, нажав кнопку Save (в меню File -> Save).

Откроем свойства устройства PC0 нажав на его изображение. Перейдем к вкладке Desktop и симулируем работу run нажав Command Prompt.

Список команд получим, если введем ? и нажмем Enter. Для конфигурирования компьютера воспользуемся командой ipconfig из командной строки, например: ipconfig 192.168.1.2 255.255.255.0

Выполним настройку всех компьютеров в соответствии с таблицей 3.

Таблица 3 – Данные для настройки конфигурации

Устройство	IP ADRESS	SUBNET MASK
PC0	192.168.1.2	255.255.255.0
PC1	192.168.1.3	255.255.255.0
PC2	192.168.1.4	255.255.255.0
PC3	192.168.1.5	255.255.255.0

IP адрес и маску сети также можно вводить в удобном графическом интерфейсе устройства (рисунок 11). Поле DEFAULT GATEWAY – адреса шлюза не важно, так как создаваемая сеть не требует маршрутизации.

Таким же путем настроим каждый компьютер.

7. На каждом компьютере посмотрим назначенные адреса командой ipconfig без параметров. Если все сделано правильно, можно выполнить ping любого узла из любого узла. Например, зайдя на компьютер PC3 и выполнив ping узла PC0, получим отчет, подобный рисунку 12.

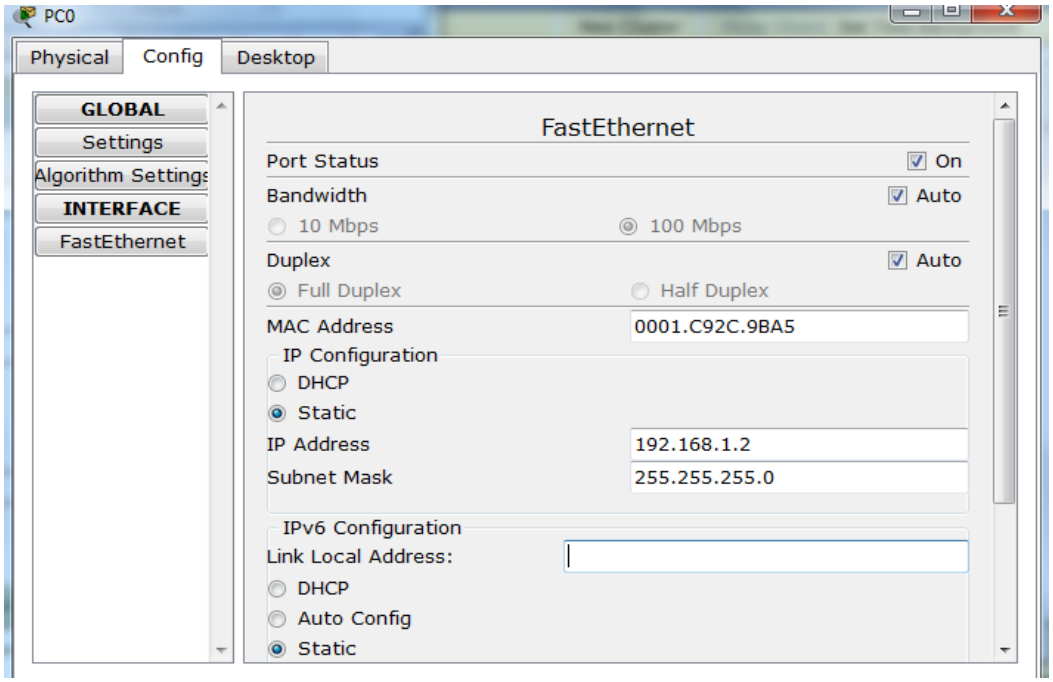


Рисунок 11 – Интерфейс настройки конфигурации устройств

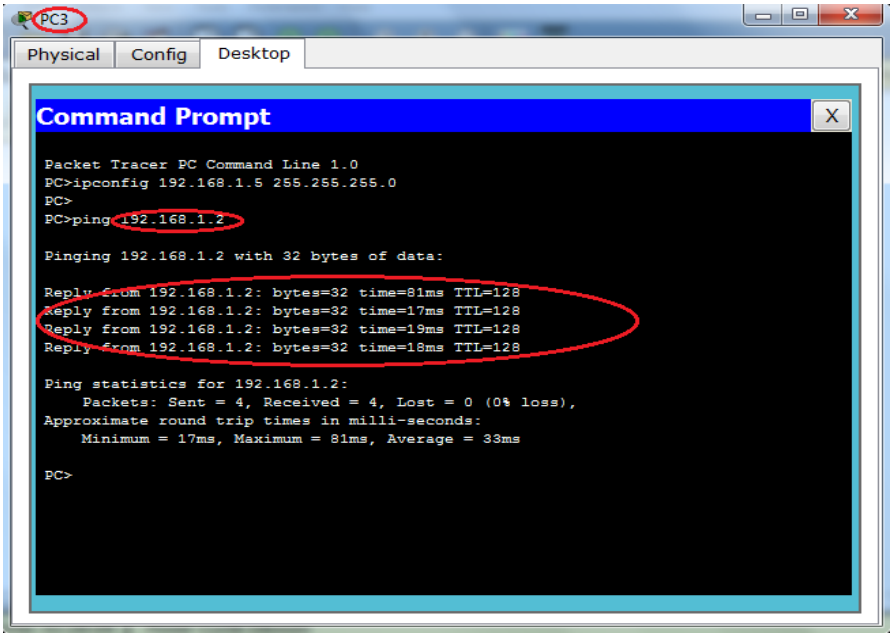


Рисунок 12 –Проверка настройки конфигурации устройств

3 Начальная конфигурация маршрутизатора Cisco

3.1 Основные понятия процесса маршрутизации

Задача маршрутизации состоит в определении последовательности узлов для передачи пакета от источника до адресата. Каждый маршрутизатор содержит таблицу сетей, подключенных локально, и интерфейсов, через которые осуществляется подключение. В таблицах маршрутизации также содержатся сведения о маршрутах или путях, по которым маршрутизатор связывается с удаленными сетями, не подключенными локально.

Эти маршруты могут назначаться администратором статически или определяться динамически при помощи программного протокола маршрутизации.

Каждый маршрутизатор принимает решения о направлении пересылки пакетов на основании таблицы маршрутизации. Таблица маршрутизации содержит набор правил. Каждое правило в наборе описывает шлюз или интерфейс, используемый маршрутизатором для доступа к определенной сети.

Маршрут состоит из четырех основных компонентов:

- значение получателя;
- маска;
- интерфейс;
- адрес шлюза;
- стоимость маршрута или метрика маршрута.

Чтобы переслать сообщение получателю, маршрутизатор извлекает IP-адрес получателя из пакета и находит соответствующее правило в таблице маршрутизации.

Значения получателей в таблице маршрутизации соответствуют адресам сетей получателей.

Чтобы определить наличие маршрута к IP-адресу получателя в таблице, маршрутизатор сначала определяет число битов, задающих адрес сети получателя.

Затем маршрутизатор ищет в таблице маску подсети, присвоенную каждому из потенциальных маршрутов. Маршрутизатор применяет каждую из масок подсети к IP-адресу получателя в пакете и сравнивает полученный адрес сети с адресами отдельных маршрутов в таблице:

- при обнаружении совпадающего адреса пакет пересылается на соответствующий интерфейс или к соответствующему шлюзу;
- если адрес сети соответствует нескольким маршрутам в таблице маршрутизации, маршрутизатор использует маршрут с наиболее точным или наиболее длинным совпадающим фрагментом адреса сети;
- иногда для одной сети адресата существует несколько маршрутов с равной стоимостью: маршрут, используемый маршрутизатором, выбирается на основе правил протокола маршрутизации;
- в отсутствие совпадающих маршрутов маршрутизатор направляет сообщение на шлюз, указанный в маршруте по умолчанию, если он настроен.

В маршрутизаторах Cisco содержимое таблицы маршрутизации можно просмотреть по команде IOS **show ip route**. В таблице маршрутизации могут содержаться маршруты нескольких типов:

Прямые маршруты. При включении питания маршрутизатора активируются настроенные интерфейсы. После выхода этих интерфейсов в рабочий режим маршрутизатор будет хранить адреса непосредственно подключенных локальных сетей в виде прямых маршрутов в таблице маршрутизации. В маршрутизаторах Cisco такие маршруты обозначаются в таблице маршрутизации префиксом C. Они автоматически обновляются при перенастройке или отключении маршрута.

Статические маршруты. Сетевой администратор может вручную настроить статический маршрут в конкретную сеть. Статические маршруты не изменяются до тех пор, пока администратор не перенастроит их вручную. В таблице маршрутизации эти маршруты обозначаются буквой S.

Динамические (динамически обновляемые) маршруты. Динамические маршруты автоматически создаются и обновляются протоколами маршрутизации. Протоколы маршрутизации реализуются в программах, которые выполняются на маршрутизаторах и осуществляют обмен сведениями о маршрутизации с другими маршрутизаторами в сети. Динамически обновляемые маршруты обозначаются в таблице маршрутизации приставкой, характеризующей тип протокола, создавшего маршрут. Например, R обозначает информационный протокол маршрутизации (RIP).

Маршрут по умолчанию. Для сетей, путь к которым отсутствует в таблице маршрутизации, используется шлюз, указанный в маршруте по умолчанию. Маршрут по умолчанию является статическим маршрутом. Обычно маршруты по умолчанию указывают на следующий маршрутизатор на пути к Интернет-провайдеру. Если в подсети присутствует только один маршрутизатор, он автоматически выбирается для маршрута по умолчанию, поскольку обмен трафиком с локальной сетью в обоих направлениях может осуществляться только через него.

3.2 Пример начальной конфигурации маршрутизатора Cisco

Исходные данные – топология сети, таблица сетевых адресов – приведены на рисунке 13 и в таблице 4.

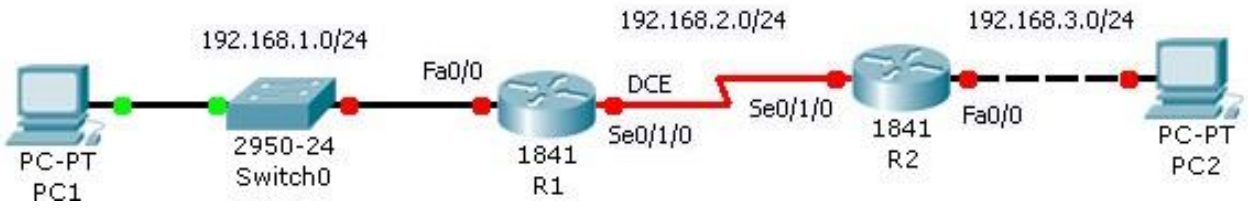


Рисунок 13 – Топология сети

Таблица 4 – Таблица сетевых адресов

Device	Interface	IP Address	Mask	Default Gateway
R1	Fa0/0	192.168.1.1	255.255.255.0	N/A
	S0/1/0	192.168.2.1	255.255.255.0	N/A
R2	Fa0/0	192.168.3.1	255.255.255.0	N/A
	S0/1/0	192.168.2.2	255.255.255.0	N/A
PC1	N/A	192.168.1.10	255.255.255.0	192.168.1.1
PC2	N/A	192.168.3.10	255.255.255.0	192.168.3.1

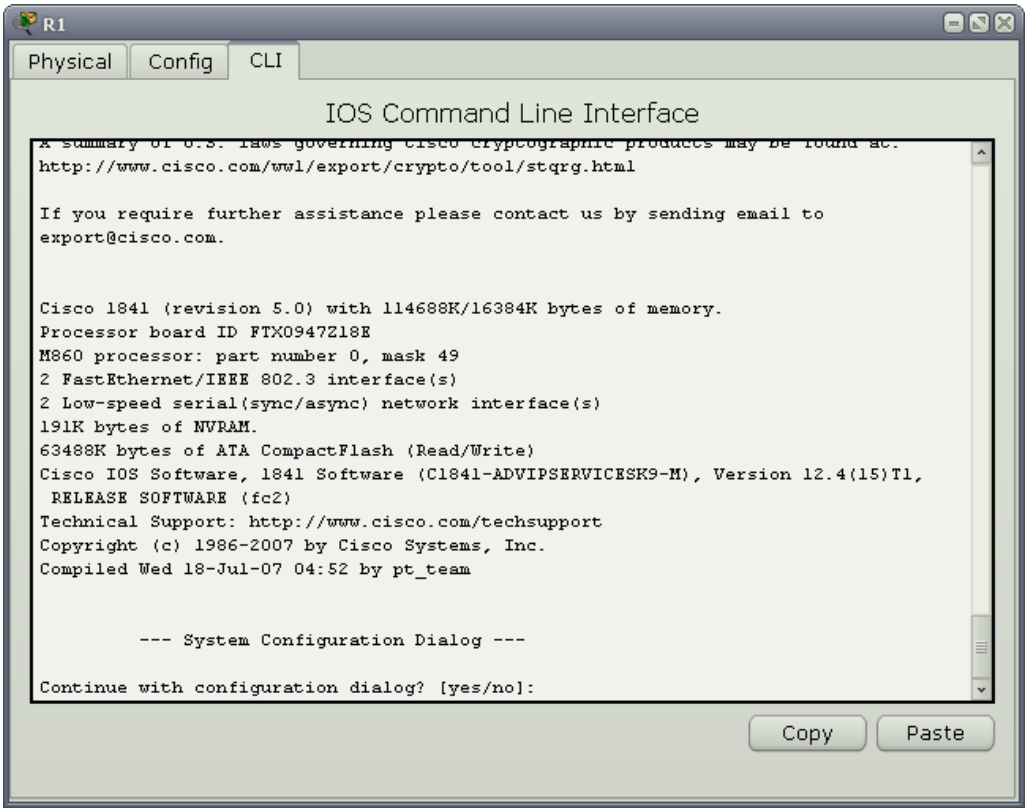
Создать (собрать и сконфигурировать) изображённую на диаграмме сеть. Настроить сетевые адреса устройств в соответствии с таблицей сетевых адресов. Произвести начальную конфигурацию маршрутизаторов. С помощью команды show и утилиты ping удостовериться, что устройства функционирую правильно.

Дальнейшие этапы выполнения работы приведены ниже.

i. Произведите начальную конфигурацию маршрутизатора R1

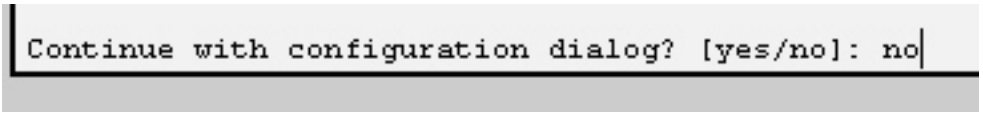
А) Двойным щелчком левой кнопки мыши откройте меню конфигурации маршрутизатора.

В) Перейдите на вкладку CLI.



С) В появившемся окне, на вопрос **“Continue with configuration dialog? [yes/no]”** ответьте нет.

D) Для этого необходимо напечатать **“no”** и нажать **Enter.**



D) Зайдите в режим **“privileged EXEC”.**

Router>**enable**
Router#

E) Зайдите в режим глобальной конфигурации маршрутизатора.

Router#**configure terminal**
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#

F) Сконфигурируйте имя маршрутизатора.

Router(config)#**hostname R1**
R1(config)#

G) Отключите **DNS lookup.**

R1(config)#**no ip domain-lookup**
R1(config)#

H) Сконфигурируйте пароль для режима **“EXEC mode”.**

R1(config)#**enable secret** _*пароль*_
R1(config)#

I) Сконфигурируйте баннер.

R1(config)#**banner motd &** _*текст*_ **&**
R1(config)#

J) Сконфигурируйте пароль, который нужно будет вводить при подключении к устройству через консоль.

R1(config)#**line console 0** R1(config-line)#**password**_*пароль*_
R1(config-line)#**login**

R1(config-line)#**exit**
R1(config)#

K) Сконфигурируйте интерфейс **FastEthernet0/0** в соответствии со схемой адресации сети.

R1(config)#**interface fastethernet 0/0**
R1(config-if)#**ip address 192.168.1.1 255.255.255.0**
R1(config-if)#**no shutdown**
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
R1(config-if)#

L) Сконфигурируйте интерфейс **Serial0/1/0** в соответствии со схемой адресации сети.

Команда **clock rate** используется для синхронизации устройств при WAN-соединениях.

R1(config-if)#**interface serial 0/1/0**
R1(config-if)#**ip address 192.168.2.1 255.255.255.0**
R1(config-if)#**clock rate 64000**
R1(config-if)#**no shutdown**
R1(config-if)#

Серийный интерфейс не активируется до тех пор, пока не будет сконфигурирован и активирован интерфейс на другой стороне. В данном случае – серийный интерфейс на маршрутизаторе R2

M) Вернитесь в режим **“privileged EXEC”.**

Use the **end** command to return to privileged EXEC mode.

R1(config-if)#**end** R1#

N) Сохраните настройки на маршрутизаторе **R1.**

*R1#**copy running-config startup-config***
Building configuration... [OK]
R1#

ii. Произведите начальную конфигурацию маршрутизатора R2

A) Для маршрутизатора R2 повторите пункты A) – G), описанные выше.

B) Сконфигурируйте интерфейс **Serial0/1/0** в соответствии со схемой адресации сети.

```
R2(config)#interface serial 0/1/0
R2(config-if)#ip address 192.168.2.2 255.255.255.0
R2(config-if)#no shutdown
%LINK-5-CHANGED: Interface Serial0/0/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0/0, changed state to up
R2(config-if)#
```

C) Сконфигурируйте интерфейс **FastEthernet0/0** в соответствии со схемой адресации сети.

```
R2(config-if)#interface fastethernet 0/0
R2(config-if)#ip address 192.168.3.1 255.255.255.0
R2(config-if)#no shutdown
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
R2(config-if)#
```

D) Вернитесь в режим **“privileged EXEC”.**

F) Use the **end** command to return to privileged EXEC mode.

```
R1(config-f)#end
R1#
```

G) Сохраните настройки на маршрутизаторе **R2.**

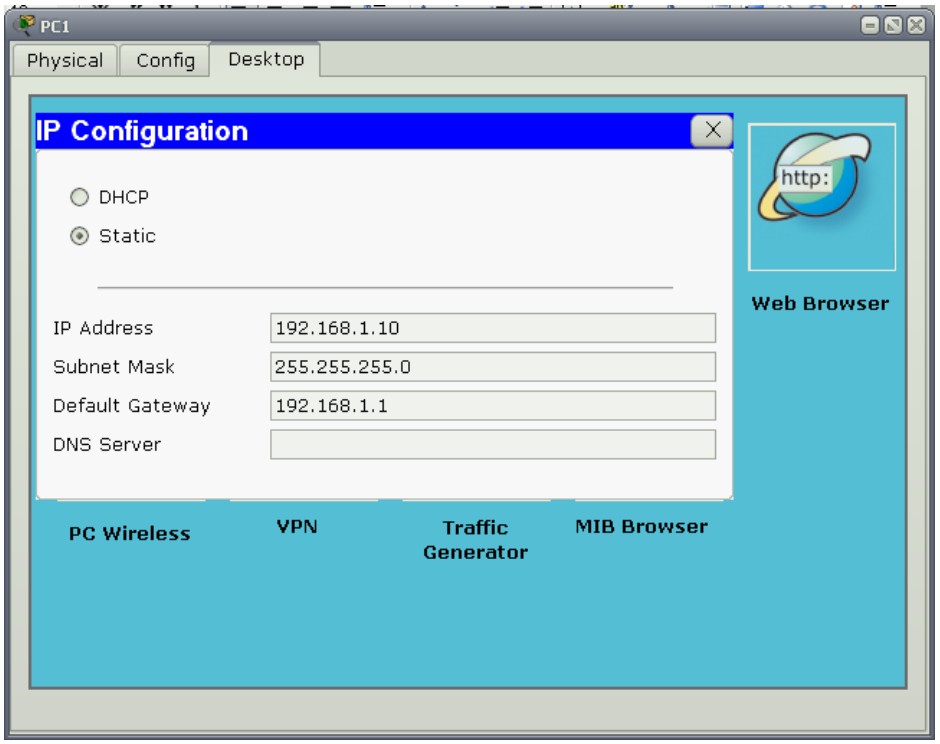
```
R1#copy running-config startup-config
Building configuration... [OK]
R1#
```

iii. Сконфигурируйте сетевые настройки на конечных устройствах

A) Двойным щелчком левой кнопки мыши откройте мену конфигурации PC1.

B) Перейдите на вкладку Desktop.

C) Нажмите на кнопку **IP configuration** и занесите необходимые параметры.



D) Повторите пункты A) – C) для PC2.

iv. Проверка и тестирование сети

A) С помощью команды **show ip route** убедитесь, что в таблицах маршрутизации присутствуют сети, в которых находятся интерфейсы маршрутизатора.

Вывод команды **show ip route** должен выглядеть следующим образом:

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.1.0/24 is directly connected, FastEthernet0/0
C    192.168.2.0/24 is directly connected, Serial0/0/0

------------------------

R2#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.2.0/24 is directly connected, Serial0/0/0
C    192.168.3.0/24 is directly connected, FastEthernet0/0
```

B) С помощью команды **show ip interface brief** убедитесь, что интерфейсы маршрутизатора настроены и активизированы.

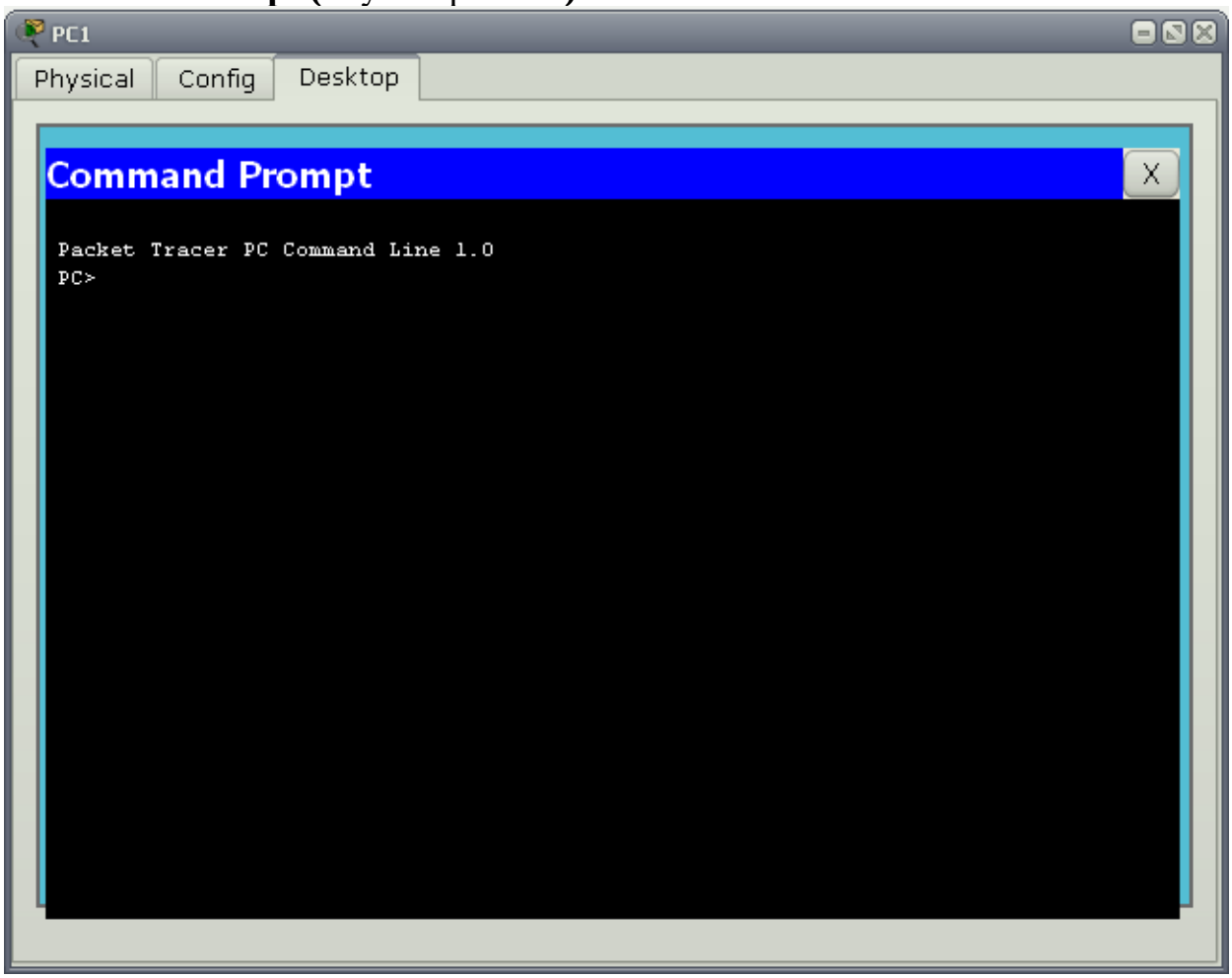
Вывод команды **show ip interface brief** должен выглядеть следующим образом:

```
R1#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.1.1     YES manual up                    up
FastEthernet0/1        unassigned      YES unset  administratively down down
Serial0/0/0              192.168.2.1     YES manual up                      up
Serial0/0/1              unassigned      YES unset  administratively down down
Vlan1                    unassigned      YES manual administratively down down


R2#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.3.1     YES manual up                    up
FastEthernet0/1        unassigned      YES unset  administratively down down
Serial0/0/0              192.168.2.2     YES manual up                      up
Serial0/0/1              unassigned      YES unset  down                    down
Vlan1                    unassigned      YES manual administratively down down
```

C) С помощью утилиты **ping** проверьте доступность устройств в сети.

Чтобы запустить утилиту **ping** на конечном устройстве (на PC) необходимо: На вкладке **Desktop** нажать на кнопку **Command Prompt (**эмулятор **CMD)**.



Используя утилиту ping, ответьте на следующие вопросы:

1. С PC1 возможно пропинговать маршрутизатор R1? Если да, то какой из интерфейсов маршрутизатора?
2. С PC2 возможно пропинговать маршрутизатор R2? Если да, то какой из интерфейсов маршрутизатора?
3. С PC2 возможно пропинговать PC1?

4 Лабораторная работа №1. Изучение пакета Cisco Packet Tracer. Начальная конфигурация маршрутизатора Cisco

Цель работы. Изучение возможностей и порядка применения пакета Cisco Packet Tracer. Приобретение навыков по начальному конфигурированию маршрутизаторов.

Порядок выполнения работы

ЧАСТЬ 1

1. Изучить: теоретический и практический материал части 1 (п.1, п.2); синтаксис сетевых утилит ipconfig, ping.

2. Выполнить в Packet Tracer практическую часть 1.

3. Получить номер собственного варианта и выполнить в Packet Tracer задание для самостоятельной работы.

4. Предъявить преподавателю результат выполнения задания для самостоятельной работы. Продемонстрировать с помощью утилиты ping правильное взаимодействие между любыми компьютерами.

Задание для самостоятельной работы

Создайте топологию – рисунок 13.

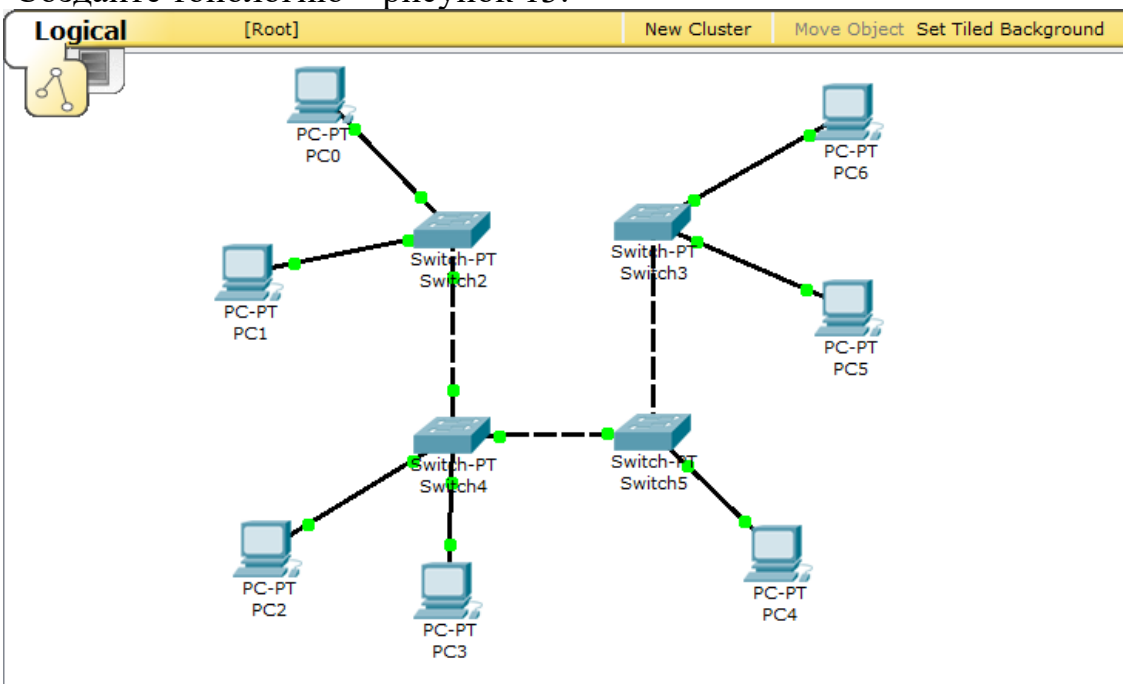


Рисунок 13 – топология сети

Назначьте компьютерам адреса, согласно варианту (v=1, 2, …), используя данные таблицы 4

Таблица 4 – Исходные данные

Устройство	IP ADDRESS	SUBNET MASK
PC1	v.1+2v.1+3v.1+5v	255.255.255.0
PC2	v.1+2v.1+3v.2+3v	255.255.255.0
PC3	v.1+2v.1+3v.3+4v	255.255.255.0
PC4	v.1+2v.1+3v.4+2v	255.255.255.0
PC5	v.1+2v.1+3v.5+3v	255.255.255.0
PC6	v.1+2v.1+3v.6+2v	255.255.255.0
PC7	v.1+2v.1+3v.7+4v	255.255.255.0

Например, для варианта 7 (v=7) и компьютера PC1 имеем IP ADDRESS 7.15.22.35

Назначьте компьютерам разные имена в командной строке.

Проверьте работоспособность сети с использованием ping.

ЧАСТЬ 2

1. Изучить теоретические указания по начальному конфигурированию маршрутизатора Cisco (п.3); создать проект приведенной топологии сети (для контроля правильности проекта допускается использовать конфигурацию lab041.pkt).

2. Модифицировать сетевые адреса устройств по правилу 192.168.x.y+v, где x, y – величины, взятые из исходного варианта топологии, v – номер индивидуального варианта студента.

3. Выполнить приведенные этапы конфигурации устройств.

4. Выполнить тестирование сети по методике, указанной в п.3.

5. Подготовиться к защите по теоретической и практической части работы.

Содержание отчета

ПО ЧАСТИ 1:

1. По п. 3 задания - скриншот топологии ЛВС.

2. По п. 4 задания – скриншоты:

- проверки сетевой конфигурации (ipconfig) - для 3-4 конечных узлов;
- проверки работоспособности сети (ping) - между 3-4 парами узлов.

Привести разъяснения относительно информации, полученной в результате работы ipconfig, ping.

ПО ЧАСТИ 2:

3. Топология сети с маршрутизаторами и сетевыми адресами согласно варианта

4. Таблица сетевых адресов.

5. Ход конфигурирования маршрутизаторов и конечных устройств по методике, приведенной в п.3.

6. Ход и результаты проверки и тестирования сети по методике, приведенной в п.3.

5 Настройка статической маршрутизации на устройствах Cisco

A) Откройте программу Packet Tracer и загрузите сетевую топологию, которая получилась в результате успешного выполнения лабораторной №4.

B) Проверьте правильность начальной конфигурации устройств:

C) Проверьте конфигурацию и активность интерфейсов маршрутизатора R1.

```
R1#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.1.1     YES manual up                    up
FastEthernet0/1        unassigned      YES manual administratively down down
Serial0/1/0            192.168.2.1     YES manual up                    up
Serial0/1/1            unassigned      YES manual administratively down down
Vlan1                  unassigned      YES manual administratively down down
```

D) Проверьте информацию в таблице маршрутизации R1.

```
R1#show ip route

Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.1.0/24 is directly connected, FastEthernet0/0
C    192.168.2.0/24 is directly connected, Serial0/1/0
```

E) Повторите пункты 2.1 и 2.2 для маршрутизатора R2 и удостоверьтесь, что:

- интерфейс fa0/0 имеет адрес 192.168.3.1 и правильно функционирует на физическом и канальном уровне;
- интерфейс s0/1/0 имеет адрес 192.168.2.2 и правильно функционирует на физическом и канальном уровне;
- в таблице маршрутизации присутствуют маршруты к сетям 192.168.2.0/24 и 192.168.3.0/24.

F) Настройте статическую маршрутизацию на устройстве R1

R1(config)#ip route 192.168.3.0 255.255.255.0 192.168.2.2

Параметры команды:

192.168.3.0	Сеть назначения
255.255.255.0	Маска сети назначения
192.168.2.2	Адрес следующего устройства

G) Настройте статическую маршрутизацию на устройстве R2

H) Настройте статическую маршрутизацию на устройстве R2, используя команду ip route со следующими параметрами:

Сеть назначения	192.168.1.0
Маска сети назначения	255.255.255.0
Адрес следующего устройства	Адрес ближайшего интерфейса маршрутизатора R1

I) Проверьте правильность конфигурации статической маршрутизации.

J) С устройства PC1 проверьте достижимость устройства PC2 с помощью команды ping. Если статическая маршрутизация настроена правильно, PC2 должно посылать эхо- ответы на эхо-запросы PC1.

```
PC>ping 192.168.3.10

Pinging 192.168.3.10 with 32 bytes of data:

Request timed out.
Reply from 192.168.3.10: bytes=32 time=125ms TTL=126
Reply from 192.168.3.10: bytes=32 time=125ms TTL=126
Reply from 192.168.3.10: bytes=32 time=110ms TTL=126

Ping statistics for 192.168.3.10:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 110ms, Maximum = 125ms, Average = 120ms
```

K) С помощью команды show ip route убедитесь, что статические маршруты появились в таблицах маршрутизации R1 и R2.

```
R2#show ip route

[output omitted]

C    192.168.1.0/24 is directly connected, FastEthernet0/0
C    192.168.2.0/24 is directly connected, Serial0/1/0
S    192.168.3.0/24 [1/0] via 192.168.2.2

[output omitted]

R1#show ip route

[output omitted]

S    192.168.1.0/24 [1/0] via 192.168.2.1
C    192.168.2.0/24 is directly connected, Serial0/1/0
C    192.168.3.0/24 is directly connected, FastEthernet0/0
```

L) Сохраните настройки на устройствах R1 и R2.

```
R1#copy running-config startup-config
Destination filename [startup-config]?
Building configuration...
[OK]
R1#
```

6 Лабораторная работа №2. Настройка статической маршрутизации на устройствах Cisco

Цель работы. Используя статическую маршрутизацию, обеспечить взаимодействие конечных устройств (PC1 и PC2). С помощью команды show и утилиты ping удостовериться, возможность взаимодействия конечных устройств обеспечена.

Исходные данные – топология сети, таблица сетевых адресов – приведены на рисунке 13 и в таблице 4.

Порядок выполнения работы

1. Изучить вышеприведенный материал по настройке статической маршрутизации; выполнить этапы настройки статической маршрутизации на устройствах Cisco. По требованию преподавателя продемонстрировать правильность настройки.
2. Собрать схему сети согласно выданному варианту задания (рисунок 14); распределить IP-адреса по аналогии с примером; составить таблицу сетевых адресов; сконфигурировать устройства.
3. Для собранной схемы сети выполнить настройку статической маршрутизации.

В отчете привести:

- схему сети с IP-адресами.
- таблицу IP-адресов.
- ход настройки статической маршрутизации по вышеприведенной методике.
- ход и результаты проверки и тестирования сети по вышеприведенной методике.

4. По требованию преподавателя продемонстрировать правильность настройки.
5. Подготовиться к защите работы.

7 Порядок настройки динамической маршрутизации с помощью протокола RIP на устройствах Cisco

Исходные данные – топология сети, таблица сетевых адресов – приведены на рисунке 15 и в таблице 5.

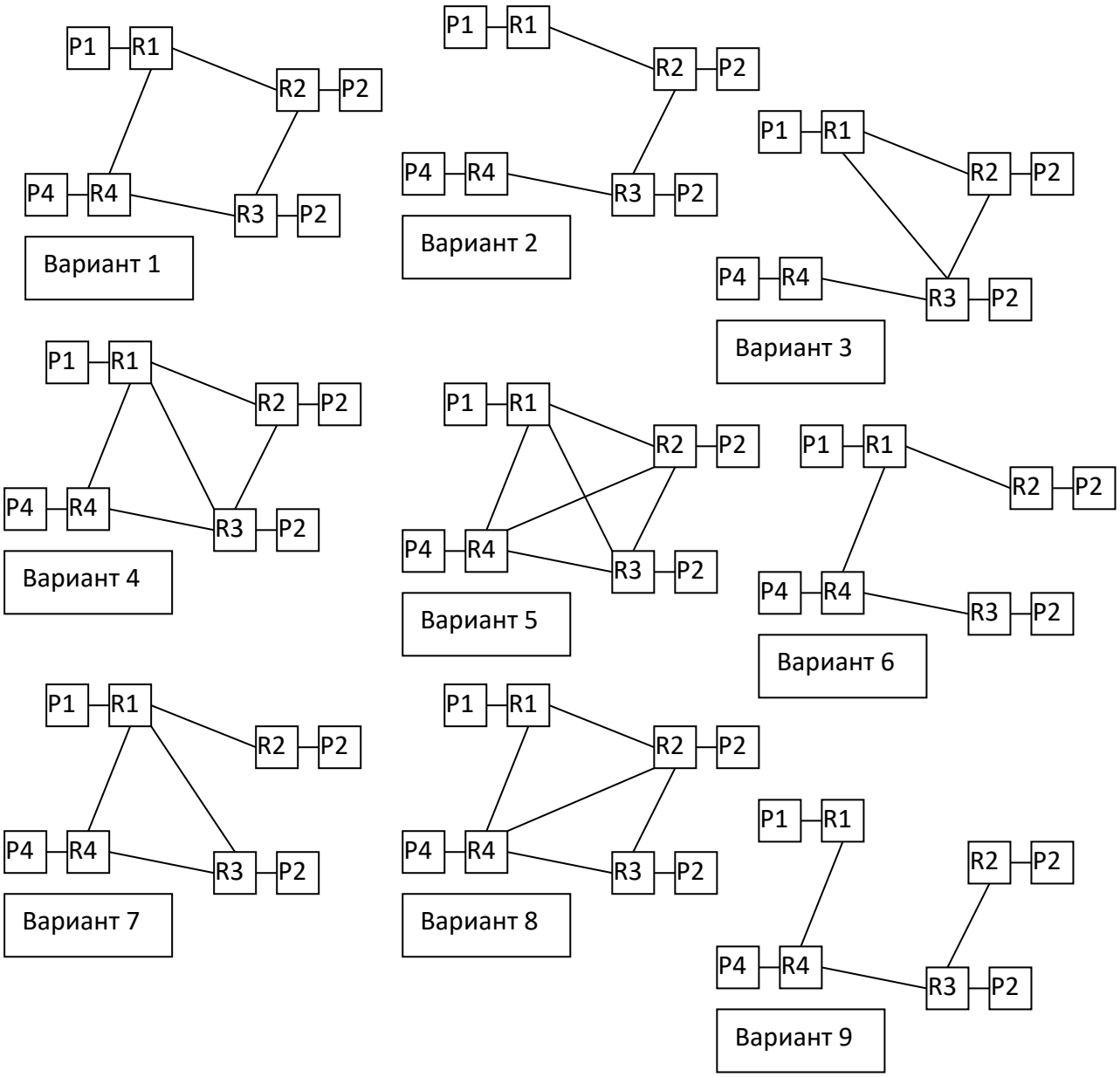


Рисунок 14 – Варианты заданий

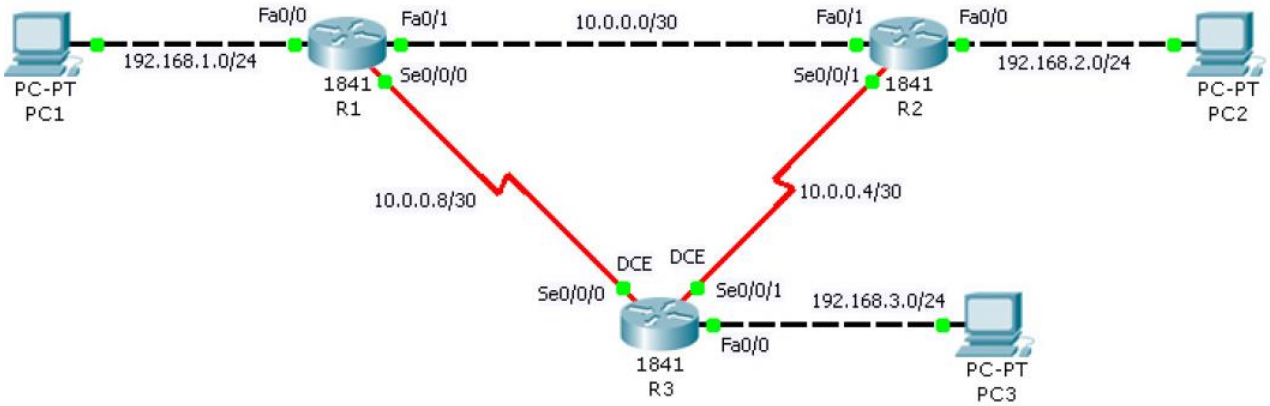


Рисунок 15 – Топология сети

Таблица 5 – Таблица сетевых адресов

Device	Interface	IP Address	Mask	Default Gateway
R1	Fa0/0	192.168.1.1	255.255.255.0	N/A
	Fa0/1	10.0.0.1	255.255.255.252	N/A
	Se0/0/0	10.0.0.9	255.255.255.252	N/A
R2	Fa0/0	192.168.2.1	255.255.255.0	N/A
	Fa0/1	10.0.0.2	255.255.255.252	N/A
	Se0/0/1	10.0.0.6	255.255.255.252	N/A
R3	Fa0/0	192.168.3.1	255.255.255.0	N/A
	Se0/0/0	10.0.0.10	255.255.255.252	N/A
	Se0/0/1	10.0.0.5	255.255.255.252	N/A
PC1	N/A	192.168.1.10	255.255.255.0	192.168.1.1
PC2	N/A	192.168.2.10	255.255.255.0	192.168.2.1
PC3	N/A	192.168.3.10	255.255.255.0	192.168.3.1

Требуется:

- настроить динамическую маршрутизацию с помощью протокола RIP на устройствах R1, R2, R3;
- обеспечить возможность взаимодействия конечных устройств PC1, PC2, PC3 между собой.

Дальнейшие этапы выполнения работы приведены ниже.

A) Откройте программу Packet Tracer и загрузите файл **lab5b.pkt.**

B) Произведите начальную конфигурацию устройств R1, R2, R3.

B.1) Откройте эмулятор командной строки.

B.2) Зайдите в режим **“privileged EXEC”.**

```
Router>enable
Router#
```

B.3) Зайдите в режим глобальной конфигурации маршрутизатора.

```
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#
```

B.4) Отключите **DNS lookup.**

```
Router(config)#no ip domain-lookup
Router(config)#
```

B.5) Сконфигурируйте имя маршрутизатора в соответствии с названиями устройств на диаграмме.

Router(config)#**hostname** *имя_маршрутизатора*

B.6) Сконфигурируйте интерфейсы в соответствии со схемой адресации.

Router(config)#**interface** *тип_интерфейса номер_интерфейса*
Router(config-if)#**ip address** *сетевой_адрес маска_сети*
Router(config-if)#**no shutdown**
Router(config-if)#**exit**
Router(config)#

Для серийных интерфейсов (Serial, Se) со стороны DCE необходимо ввести команду:

Router(config-if)#**clock rate 64000**

C) Проверьте правильность начальной конфигурации устройств в помощью команд

C.1) В помощью команды **show ip interface brief,** проверьте адреса на интерфейсах настроены правильно, и что интерфейсы функционируют на физическом и канальном уровнях.

Router(config)#**exit**
R1#**show ip interface brief**

C.2) В помощью команды **show ip route** убедитесь, что каждый маршрутизатор видит все присоединённые к нему сети.

R1#**show ip route**

D) Сконфигурируйте сетевые интерфейсы конечных устройств (PC1, PC2, PC3) в соответствии со схемой адресации сети.

E) Настройте протокол RIP на маршрутизаторах R1, R2, R3

E.1) Зайдите в режим конфигурации протокола маршрутизации

Router(config)#**router rip**
Router(config-router)#

E.2) Определите сети, которые должны передаваться по протоколу маршрутизации и интерфейсы, которые должны участвовать в обмене информацией между маршрутизаторами.

Router(config-router)#**network** *адрес_сети*
Для R1:

Router(config-router)#**network 192.168.1.0**
Router(config-router)#**network 10.0.0.0**

Для R2:

Router(config-router)#**network 192.168.2.0**
Router(config-router)#**network 10.0.0.0**

Для R3:

Router(config-router)#**network 192.168.3.0**
Router(config-router)#**network 10.0.0.0**

F) Проверка правильности работы протокола RIP.

Если протокол маршрутизации настроен правильно, то каждый маршрутизатор должен знать путь до каждой сети. Проверить этот факт можно с помощью команды **show ip route.**

R1#**show ip route**

```
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter
area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

     10.0.0.0/30 is subnetted, 3 subnets
C       10.0.0.0 is directly connected, FastEthernet0/1
R       10.0.0.4 [120/1] via 10.0.0.2, 00:00:16, FastEthernet0/1
                 [120/1] via 10.0.0.10, 00:00:20, Serial0/0/0
C       10.0.0.8 is directly connected, Serial0/0/0
C    192.168.1.0/24 is directly connected, FastEthernet0/0
R    192.168.2.0/24 [120/1] via 10.0.0.2, 00:00:16, FastEthernet0/1
R    192.168.3.0/24 [120/1] via 10.0.0.10, 00:00:20, Serial0/0/0
R1#
```

На маршрутизаторах R2 и R3 так же должны присутствовать записи для сетей: 192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, 10.0.0.0/30, 10.0.0.4/30, 10.0.0.8/30.

G) Сохраните конфигурацию устройств.

```
Router#copy running-config startup-config
Destination filename [startup-config]?
Building configuration...[OK]
Router#
```

8 Лабораторная работа №3. Настройка динамической маршрутизации с помощью протокола RIP на устройствах Cisco

Цель работы. Настроить динамическую маршрутизацию с помощью протокола RIP на устройствах R1, R2, R3; обеспечить возможность взаимодействия конечных устройств PC1, PC2, PC3 между собой.

Исходные данные – топология сети, таблица сетевых адресов – приведены на рисунке 15 и в таблице 5.

1. Изучить вышеприведенный материал; выполнить этапы настройки динамической маршрутизации с помощью протокола RIP на устройствах Cisco, изложенные в документе. По требованию преподавателя продемонстрировать правильность настройки.
2. Собрать схему сети согласно выданному варианту задания (рисунок 16); распределить IP-адреса по аналогии с вышеприведенным примером; составить таблицу сетевых адресов; сконфигурировать устройства.
3. Для собранной схемы сети выполнить настройку динамической маршрутизации с помощью протокола RIP.

В отчете привести:

- схему сети с IP-адресами.
- таблицу IP-адресов.
- ход настройки маршрута по протоколу RIP по вышеприведенной методике.
- ход и результаты проверки и тестирования сети вышеприведенной методике.

4. По требованию преподавателя продемонстрировать правильность настройки.
5. Подготовиться к защите работы.

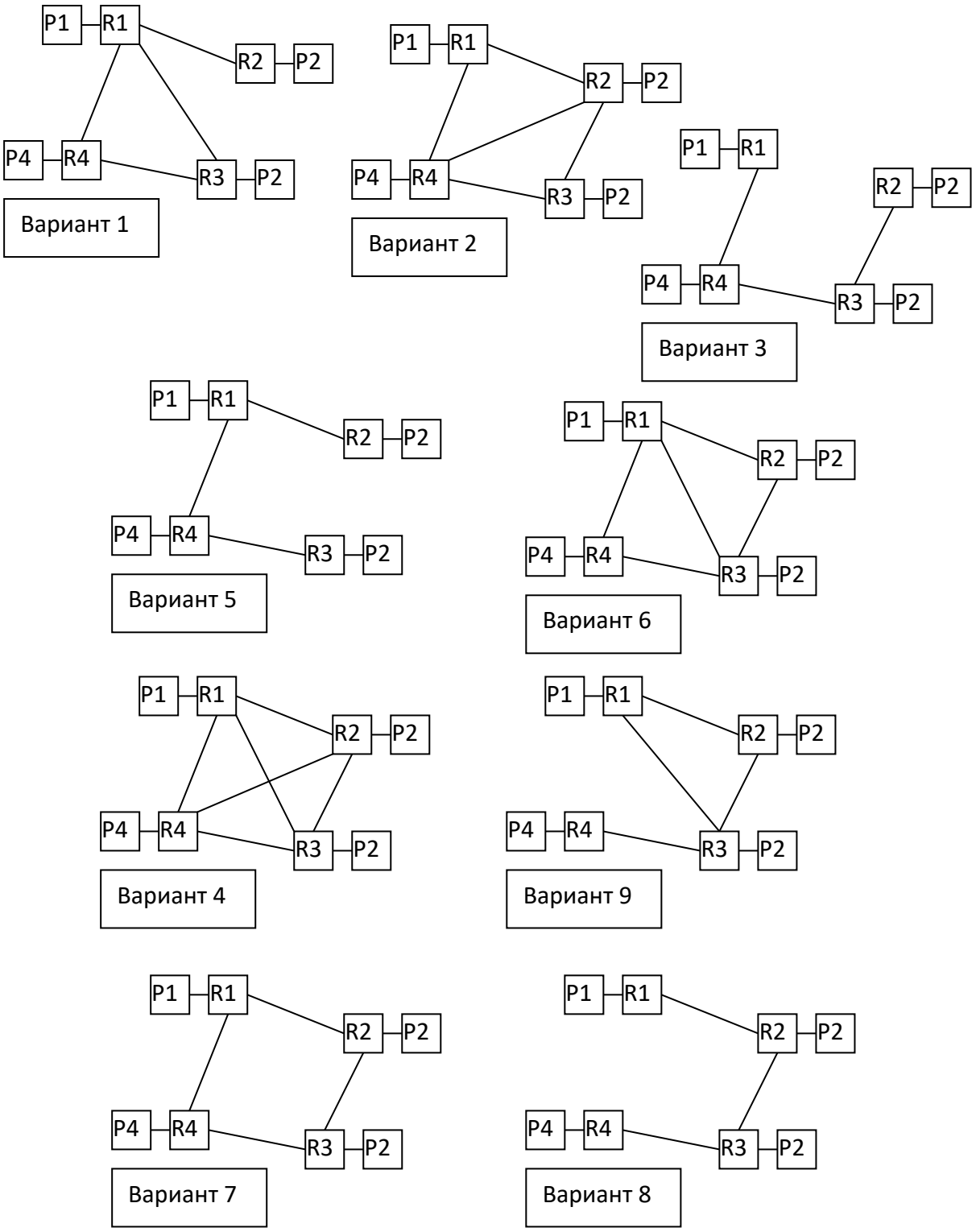


Рисунок 16 – Варианты заданий

Список использованных источников

1. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для ВУЗов. Юбилейное издание. – СПб.: Питер, 2020. – 1008 с.

2. Уколов С.С., Таваева А.Ф. Сети и системы телекоммуникаций. – Екатеринбург, 2018. – 63 с.

3. Таненбаум, Э. Компьютерные сети / Э. Таненбаум; Пер. с англ. А. Леонтьева. – 3-е изд. – М.; СПб.; Н. Новгород и др. : Питер, 2002. – 846 с.

2.5 УЧЕБНО-МЕТОДИЧЕСКИЙ МАТЕРИАЛ ДЛЯ ПРОВЕДЕНИЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

ОГЛАВЛЕНИЕ

1 Практическая работа №1. Дисциплины обслуживания заявок в компьютерных системах

1.1 Характеристики бесприоритетных дисциплин обслуживания

Под дисциплиной обслуживания заявок понимаются правила, по которым заявки выбираются из очередей для обслуживания.

При бесприоритетной дисциплине заявки разных типов не имеют заранее определенных привилегий на досрочное обслуживание. Это правило выполняется, если заявки на обслуживание выбираются:

- в порядке поступления (первой обслуживается заявка, поступившая раньше других) – FIFO;
- в порядке, обратном порядку поступления заявок (первой обслуживается заявка, поступившая позже других) – LIFO;
- наугад, т.е. путем случайного выбора из очереди.

Эти три бесприоритетных дисциплины характеризуются одинаковым средним временем ожидания заявок, но дисциплина FIFO минимизирует дисперсию времени ожидания, т.е. уменьшает разброс времени ожидания относительно среднего значения. По этой причине дисциплина FIFO используется наиболее часто.

Бесприоритетное обслуживание заявок на основе дисциплины FIFO организуется в соответствии с рисунком 1, где *Пр* – процессор (обслуживающее устройство) и *О* – очередь заявок типа $z_1...z_M$. Вновь поступившая заявка заносится в конец очереди. Заявки выбираются на обслуживание из начала очереди.

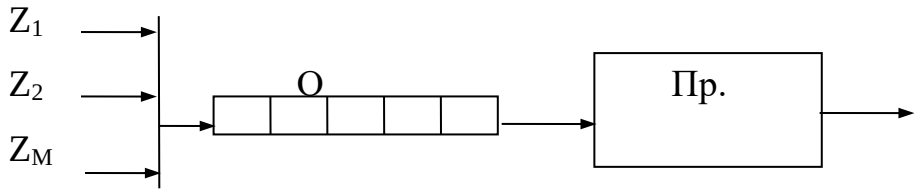


Рисунок 1 – Организация бесприоритетного обслуживания заявок на основе дисциплины FIFO

Пусть в систему поступают заявки M типов с интенсивностями $\lambda_1,..,\lambda_M$. Предположим, что каждый из входящих потоков заявок – пуассоновский (см. п.1.4). В таком случае суммарный поток заявок также пуассоновский и его интенсивность равна:

$$\Lambda=\sum_{i=1}^{M}\lambda_i \qquad (1.1)$$

Пусть известны также математические ожидания $v_1,...,v_M$ и вторые начальные моменты $v^{(2)}_1,...,v^{(2)}_M$ времени обслуживания заявок типа *1,...,M*

соответственно. Эти значения характеризуют распределение времени выполнения соответствующих программ. Тогда при использовании бесприоритетной дисциплины обслуживания среднее время ожидания заявок всех типов одинаково и равно

$$w = \frac{\sum_{i=1}^{M} \lambda_i v_i^{(2)}}{2(1-R)} \qquad (1.2)$$

где $R=(\rho_1+...+\rho_M)<1$ - суммарная загрузка системы и

$$\rho_i = \lambda_i v_i.$$

Выразим второй начальный момент $v_i^{(2)}$ через коэффициент вариации $s_i=\sigma_i/v_i$, определяющий отношение среднеквадратичного отклонения длительности обслуживания σ_i к его математическому ожиданию: $v_i^{(2)}=v_i^2+\sigma_i^2=v_i^2(1+s_i^2)$. С учетом этого среднее время ожидания

$$w = \frac{\sum_{i=1}^{M} \rho_i v_i (1+s_i^2)}{2(1-R)} \qquad (1.3)$$

Среднее время обслуживания

$$v_i = \theta_i / V,$$

где θ_i – трудоемкость выполнения заявки типа i, V – быстродействие процессора.

Из (1.3) видно, что среднее время ожидания заявок в очереди минимально при постоянной длительности обслуживания заявок каждого типа ($\upsilon_i=0$) и увеличивается по мере роста дисперсии времени обслуживания. Среднее время ожидания существенно зависит от общей загрузки R процессора Пр. При $R\to1$ время ожидания заявок стремится к $+\infty$, т.е. заявки могут ожидать обслуживания сколь угодно долго. При постоянных характеристиках входящих потоков увеличение общей загрузки R соответствует уменьшению быстродействия процессора Пр.

Время пребывания заявки типа $i=1,..,M$ в системе равно сумме времени ожидания w_i и времени обслуживания v_i, т.е.

$$u_i = w_i + v_i.$$

Поскольку при бесприоритетном обслуживании $w_i = w$, то время пребывания $u_i = w + v_i$. При одинаковом времени ожидания время пребывания заявок разных типов различно.

1.2 Характеристики дисциплины обслуживания с относительными приоритетами заявок

Если требуется, чтобы заявки некоторого типа имели меньшее время ожидания (время пребывания), чем заявки других типов, то необходимо первым предоставить преимущественное право на обслуживание, называемое *приоритетом*.

В сетях связи для ЭВМ характерной является передача сообщений с различными приоритетами. Коротким сообщениям, содержащим подтверждения, часто назначают более высокий приоритет, чем информационным сообщениям. По сети могут передаваться сообщения 2 и более категорий срочности.

Если приоритеты учитываются только в момент выбора заявки на обслуживание, то их называют *относительными*. В момент выбора сравниваются приоритеты заявок, находящихся в состоянии ожидания, их обслуживание предоставляется заявке с наиболее высоким приоритетом, после этого выбранная заявка захватывает процессор. Если в процессе обслуживания той заявки поступают заявки с более высокими приоритетами, то процесс обслуживания заявки, имеющей больший приоритет, не прекращается, т.е. эта заявка, захватив процессор, оказывается наиболее приоритетной. Этот возникший приоритет относителен: он имеет место только после захвата процессора. При использовании относительных приоритетов обработка заявок организуется по схеме на рисунке 2.

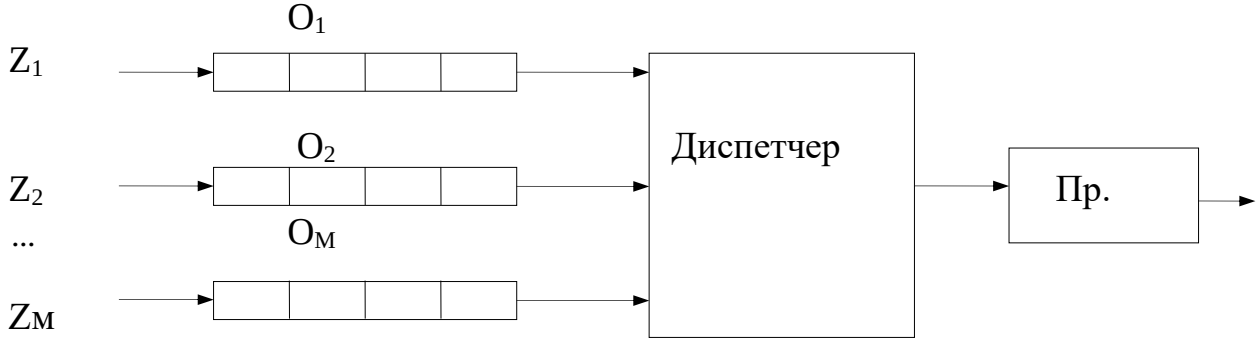


Рисунок 2 – Организация обработки заявок с относительными приоритетами

Заявкам типа $Z_1..Z_M$ присвоены относительные приоритеты *1, ..., M* соответственно. Заявка Z_P , поступившая в систему, заносится в очередь O_p, в которой хранятся заявки приоритета $p=1, .., M$. В очереди O_p заявки упорядочены по времени поступления. Когда процессор заканчивает ранее начатое обслуживание, то управление передается программе *диспетчер*. Диспетчер выбирает на обслуживание заявку с наибольшим приоритетом - заявку Z_i, если очереди $O_1, .., O_{i-1}$ не содержат заявок. Выбранная заявка захватывает процессор на все время обслуживания.

Если в систему поступает M простейших потоков с интенсивностями $\lambda_1,..,\lambda_M$ и длительности обслуживания заявок каждого потока имеют математические ожидания $v_1,...,v_M$ и вторые начальные моменты $v^{(2)}_1,...,v^{(2)}_M$ соответственно, то среднее время ожидания заявок, имеющих приоритеты $k = 1,..,M$, определяется значениями:

$$w_k = \frac{\sum_{i=1}^{M} \rho_i v_i (1 + s_i^2)}{2(1 - R_{k-1})(1 - R_k)} \qquad (1.3)$$

где $R_{k-1} = \rho_1 + \rho_2 + ... + \rho_{k-1}$ *и* $R_k = \rho_1 + \rho_2 + ... + \rho_k$ - загрузки, создаваемые потоками заявок $Z_1..Z_{k-1}$ *и* $Z_1..Z_k$ соответственно. Введение относительных приоритетов приводит к уменьшению времени ожидания заявок с высокими приоритетами и увеличению времени ожидания заявок с низкими приоритетами по сравнению с бесприоритетным обслуживанием.

1.3 Характеристики дисциплин обслуживания с абсолютными приоритетами

В ряде случаев время ожидания заявок некоторых типов нужно уменьшить в такой степени, которая недостижима при использовании относительных приоритетов. Время ожидания значительно уменьшиться, если при поступлении высокоприоритетной заявки обслуживание ранее поступившей заявки с низшим приоритетом прерывается и процессор тут же предоставляется для обслуживания высокоприоритетной заявки. Дисциплина обслуживания, при которой высокоприоритетная заявка прерывает обслуживание заявки с низким приоритетом, называется *дисциплиной обслуживания с абсолютными приоритетами.* При использовании абсолютных приоритетов обслуживание заявок организуется по схеме рисунка 3. Для каждого потока заявок $Z_1..Z_M$ организуется очередь $O_1, .., O_M$, в которой заявки размещаются в порядке поступления. Заявкам $Z_1..Z_M$ соответствуют абсолютные приоритеты 1,...,M. Если процессор занят обслуживанием заявки Z_i и на вход поступает заявка типа Z_j, то при $i \le j$ заявка Z_j заносится в конец очереди O_j, а при $i > j$ обслуживание заявки Z_i прерывается, заявка Z_i заносится в начало очереди O_i и диспетчер переключает прибор на обслуживание заявки Z_j.

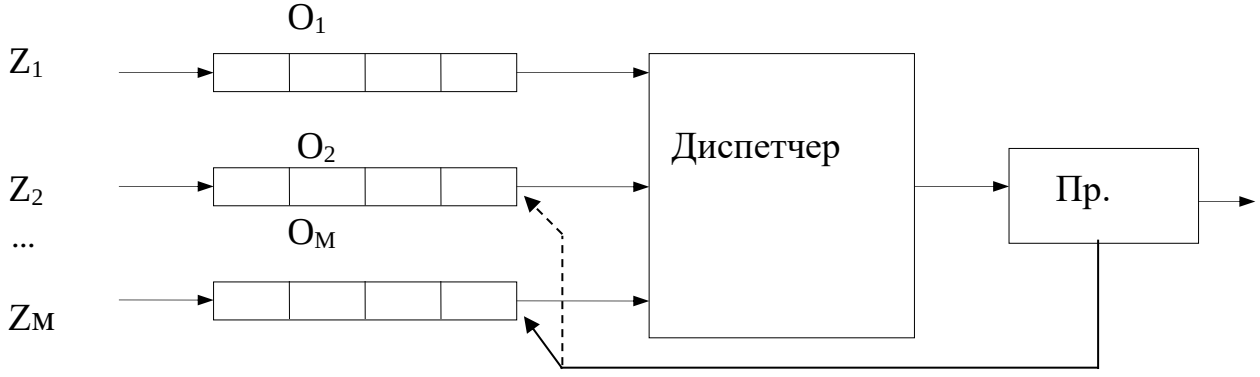


Рисунок 3 – Организация обработки заявок с абсолютными приоритетами

Обслуживание прерванных заявок может проводиться от начала; от момента прерывания (дообслуживание).

По возможности стремятся использовать второй способ – дообслуживание прерванных заявок. В случае, когда длительность обслуживания распределена по экспоненциальному закону, среднее время дообслуживания совпадает со средним временем обслуживания заявки. Когда прерывание предполагает дообслуживание, сохраняется вся информация о процессе обслуживания, необходимая для возобновления (продолжения) обслуживания. Если потоки заявок - простейшие с интенсивностями $\lambda_1,..,\lambda_M$ и длительности обслуживания заявок каждого потока имеют математические ожидания $v_1,...,v_M$ и вторые начальные моменты $v^{(2)}_1,...,v^{(2)}_M$ и прерванные заявки дообслуживаются от точки прерывания ,то среднее время ожидания заявки с абсолютным приоритетом $k = 1,..,M,$ определяется значением:

$$w_k = \frac{R_{k-1} v_k}{1 - R_{k-1}} + \frac{\sum_{i=1}^{k} \rho_i v_i \,(1 + s_i^2)}{2(1 - R_{k-1})(1 - R_k)} \qquad (1.4)$$

Присваивание заявкам абсолютных приоритетов приводит к уменьшению времени ожидания заявок с высокими приоритетами, но одновременно с этим увеличивается время ожидания низкоприоритетных заявок.

1.4 Справочная информация: потоки заявок, простейший поток

Потоки заявок. Совокупность событий, распределенных во времени, называется *потоком заявок.* Различают *входящие* и *выходящие потоки заявок,* которые поступают в систему и соответственно покидают ее.

В общем случае поток заявок рассматривается как случайный процесс, задаваемый функцией распределения промежутков времени между моментами поступления двух соседних заявок. Важнейшая характеристика потока — его *интенсивность* λ, равная среднему числу заявок, поступающих в единицу времени. Величина $1/\lambda$, обратная интенсивности, определяет средний интервал времени между двумя последовательными заявками.

Стационарные и нестационарные потоки заявок. Поток заявок может быть стационарным и нестационарным: *стационарным* — если его характеристики не изменяются во времени; *нестационарным* — если характеристики изменяются во времени.

Характеристики ЦУС определяются наиболее просто для стационарного режима функционирования системы, предполагающего стационарность потоков заявок. По этой причине нестационарные потоки аппроксимируются на отдельных отрезках времени стационарными.

Простейший поток. В теории массового обслуживания наибольшее число аналитических результатов получено для потока, называемого простейшим.

Простейший поток обладает следующими свойствами: *стационарностью* (вероятностные характеристики потока не зависят от времени); *отсутствием последействия* (заявки поступают в систему независимо друг от друга, в частности длина интервала времени до момента поступления следующей заявки не зависит от

того, поступила в начальный момент заявка или нет); *ординарностью* (в каждый момент времени в систему может поступить не более одной заявки). Для простейшего потока интервалы времени между двумя последовательными заявками—независимые случайные величины с функцией распределения

$$P(\tau) = 1 - e^{-\lambda\tau} \tag{1.5}$$

Распределение такого вида называется *показательным (экспоненциальным)* и имеет плотность

$$p(\tau) = \lambda \cdot e^{-\lambda\tau} \tag{1.6}$$

Математическое ожидание длины интервала времени между последовательными моментами поступления заявок

$$E(\tau) = \int_0^\infty \tau \cdot p(\tau) d\tau = 1/\lambda \tag{1.7}$$

Дисперсия интервала времени между последовательными моментами поступления заявок

$$D(\tau) = \int_0^\infty \tau \cdot p(\tau) d\tau - (E[\tau])^2 = 1/\lambda^2 \tag{1.8}$$

Вычислим вероятность появления коротких интервалов между двумя последовательными заявками, длина которых меньше математического ожидания $E[\tau]=1/\lambda$:

$$\Pr(\tau < E(\tau)) = \int_0^{1/\lambda} \tau \cdot p(\tau) d\tau = \lambda \int_0^{1/\lambda} e^{-\lambda\tau} d\tau = 1 - 1/e \approx 0.63 \tag{1.9}$$

Таким образом, короткие интервалы более часты, чем длинные, т. е. при простейшем потоке заявки обнаруживают тенденцию к группировке, что создает более тяжелые условия при работе системы по сравнению с другими распределениями потоков заявок.

Для простейшего потока число заявок, поступающих в систему за промежуток времени τ, распределено по *закону Пуассона:*

$$\Pr(k, \tau) = \frac{(\lambda \cdot \tau)^k}{k!} e^{-\lambda\tau}, (\lambda > 0) \tag{1.10}$$

где Pr *(k,* τ)—вероятность того, что за время τ в систему поступит точно *k* заявок; λ — интенсивность потока заявок.

Математическое ожидание и дисперсия распределения Пуассона:

$$\begin{aligned} E[k] &= \sum_{k=1}^{\infty} k \Pr(k, \tau) = \lambda\tau \\ D[k] &= E[k^2] - (E[k])^2 = \lambda\tau \end{aligned} \tag{1.11}$$

Распределение Пуассона (1.6) дискретно, в то время как распределение интервалов времени в простейшем потоке (1.2) непрерывно.

Если нестационарный поток, интенсивность которого функция времени $\lambda=\lambda(t)$, описывается законом распределения Пуассона, то такой поток называется *пуассоновским,* но не простейшим, поскольку не выполняется свойство стационарности, присущее простейшему потоку. Иными словами, простейший поток — это стационарный пуассоновский поток.

Простейший поток обладает следующими особенностями:

1. Сумма *M* независимых, ординарных, стационарных потоков заявок с интенсивностями $\lambda i(i=l, ..., M)$ сходится к простейшему потоку с интенсивностью

$$\Lambda = \sum_{i=1}^{M} \lambda_i$$

при условии, что складываемые потоки оказывают более или менее одинаково малое влияние на суммарный поток. Сходимость суммарного потока к простейшему осуществляется очень быстро. Практически можно считать, что сложение четырех-пяти стационарных, ординарных, независимых потоков, сравнимых по интенсивности, достаточно для того, чтобы суммарный поток был близок, к простейшему.

Таким образом, для выяснения всех свойств суммарного потока достаточно знать лишь интенсивности суммируемых потоков и практически не требуется знать внутреннюю структуру этих потоков.

1. Простейший поток обладает устойчивостью, состоящей в том, что при суммировании независимых простейших потоков получается снова простейший поток, причем интенсивности складываемых потоков суммируются.

2. Поток заявок, полученный путем случайного разрежения исходного потока, когда каждая заявка с определенной вероятностью p исключается из потока независимо от того, исключены другие заявки или нет, образует простейший поток с интенсивностью $\lambda_p = p\lambda$, где λ—интенсивность исходного потока. В отношении исходного потока заявок делается предположение лишь об ординарности и стационарности.

3. Для простейшего потока характерно, что поступление заявок через короткие промежутки времени более вероятно, чем через длинные,—63% промежутков времени между заявками имеют длину, меньшую среднего периода E[τ]=1/λ,. Следствием этого является то, что простейший поток по сравнению с другими видами потоков создает наиболее тяжелый режим работы системы. Поэтому предположение о том, что на вход системы поступает простейший поток заявок, приводит к определению предельных значений характеристик качества обслуживания. Если реальный поток отличен от простейшего, то система будет функционировать не хуже, чем это следует из полученных оценок.

4. Интервал времени между произвольным моментом времени и моментом поступления очередной заявки имеет такое же распределение *(1.1)* с тем же средним E[τ]=1/λ, что и интервал времени между двумя последовательными заявками. Эта особенность простейшего потока является следствием отсутствия последействия.

1.5 Порядок проведения практической работы

1.5.1 Изучить основные теоретические сведения (п.1-п.4) о потоках и дисциплинах обслуживания заявок в ВС, разъяснения преподавателя.

1.5.2 Получив вариант задания, определить времена ожидания на обслуживание для каждой из дисциплин обслуживания заявок.

1.5.3. По результатам расчета сделать выводы о влиянии типа дисциплины обслуживания и номера приоритета на времена ожидания заявок в очереди.

2 Практическая работа №2. Критерии эффективности цифровых управляющих систем

2.1 Общие положения

Цифровые управляющие системы (ЦУС) функционируют в реальном масштабе времени (согласованно с темпом поступления заявок на решение определенных задач). Это означает, что заявки должны обслуживаться системой за какое-то ограниченное время, определяемое назначением, т. е. сферой применения ЦУС. Выполнение ограничений на время обслуживания заявок — основная задача, возникающая при проектировании ЦУС. Форма постановки такого рода задач и подход к их решению зависят от вида ограничений на времена решения задач, предопределяемых назначением системы.

ЦУС в зависимости от требований к временным характеристикам принято разделять на следующие основные классы: 1) с неограниченным временем пребывания заявок; 2) с относительными ограничениями, на времена пребывания заявок; 3) с абсолютными ограничениями на времена пребывания заявок.

Рассмотрим особенности функционирования систем разных классов, предполагая, что ЦУС обслуживает заявки типа *M,* поступающие в систему с интенсивностями *λ1, ..., λм,* и что известны:

1) полное время ожидания $W_1, ..., W_м$ заявок типов 1, ..., *M,* включающее в себя время ожидания начала обслуживания и время ожидания в прерванном состоянии;

2) время пребывания $U_1, ..., U_м$ заявок в системе;

3) вероятность превышения допустимого времени ожидания $P(W_i > W_i^*)$ или пребывания заявок в системе $P(U_i > U_i^*)$, $i=1, ..., M,$ где *Wi* и Ui**—предельные ограничения соответственно на времена ожидания и пребывания заявок в системе;

4) коэффициент простоя процессора η=1—ρ.

Время пребывания и время ожидания заявки в системе связаны между собой соотношением

$$U = W + \nu \text{ г}$$

де ν—длительность обслуживания заявки в изоляции. Таким образом, эти понятия эквивалентны и могут использоваться в равной степени, поскольку время пребывания всегда может быть сведено к времени ожидания и наоборот. Ввиду, того что математические зависимости получаются более простыми для времени ожидания, в дальнейшем все выкладки будут выполняться для этой характеристики. Аналогичные зависимости для времени пребывания могут быть легко получены путем замены в этих выражениях $W = U - \nu$.

2.2 Критерии эффективности ЦУС с неограниченным временем пребывания заявок

В *ЦУС с неограниченным временем пребывания заявок* ограничения на времена пребывания заявок в явном виде не устанавливаются, однако считается, что чем дольше заявки пребывают в системе, тем ниже качество функционирования последней, т. е. тем в меньшей мере система соответствует своему назначению. Потеря качества функционирования из-за задержки обслуживания заявок характеризуется функцией штрафа

$$C_w = \sum_{i=1}^{M} \alpha_i \lambda_i w_i \qquad (2.1)$$

где C_w — штраф за задержку одной заявки типа i на единицу времени, λ_i и w_i — интенсивность потока и среднее время ожидания заявок.

Произведение $\alpha_i \lambda_i w_i$ определяет штраф за задержку обслуживания заявок типа *i,* поступающих в систему за единицу времени, а значение C_w —штраф за задержку всех заявок. Критерий (2.1) — инверсный: с увеличением качества системы значение C_w, уменьшается.

Если штрафы α_i одинаковы для всех потоков заявок, то критерий (1.1) принимает вид

$$C_w = \sum_{i=1}^{M} \lambda_i w_i$$

где $\lambda_i w_i$ — средняя длина очереди заявок типа *i.*

Значение C_w есть средняя длина очереди всех заявок.

Задержки, т. е. времена ожидания $W_1, ..., W_м$, зависят от двух факторов: быстродействия процессора и дисциплины обслуживания заявок. Увеличение быстродействия приводит к уменьшению всех этих значений за счет использования приоритетных дисциплин обслуживания можно уменьшить времена ожидания, заявок одних типов, но при этом времена ожидания заявок других типов увеличатся.

Рассмотрим вопрос о возможности использования функции (2.1) в качестве критерия эффективности систем с неограниченным временем пребывания заявок. Допустим, что функция *Cw* используется в качестве критерия при решении задачи выбора быстродействия процессора. С увеличением быстродействия процессора уменьшаются значения $W_1, ..., W_м$, и, следовательно, уменьшается значение *Cw.* Максимальная эффективность системы достигается при бесконечном быстродействии процессора, что свидетельствует о некорректности поставленной задачи. Таким образом, использование критерия (2.1) предполагает, что быстродействие процессора задано.

Критерий (2.1) может быть использован в задаче выбора дисциплины

обслуживания заявок. Действительно, различным дисциплинам соответствуют различные значения времени ожидания W_1, ..., $W_м$, которые при изменении дисциплины взаимно перераспределяются в соответствии с законом сохранения времени ожидания. Дисциплина обслуживания заявок может считаться оптимальной для определенной ЦУС, если ей соответствует минимальное значение *Cw,* по сравнению с другими дисциплинами обслуживания.

При использовании критерия (2.1) выбор оптимальной дисциплины обслуживания может проводиться в принципе для любого значения быстродействия процессора ЦУС, обеспечивающего существование стационарного режима в системе, поскольку это не сказывается на выборе оптимальной дисциплины. Минимальное значение быстродействия процессора, при котором может быть найдена дисциплина обслуживания, удовлетворяющая ограничениям на время ожидания заявок, будем рассматривать как *нижнюю оценку быстродействия процессора.*

После того как найдена оптимальная дисциплина обслуживания, возникает задача уточнения быстродействия процессора, заключающаяся в определении оптимального значения в смысле некоторого критерия. Для построения такого критерия будем рассуждать следующим образом. С целью уменьшения времени ожидания заявок в системе необходимо иметь процессор с высоким быстродействием. Но с увеличением быстродействия процессора растет коэффициент *его* простоя и в пределе приближается к единице:

$$\eta = 1 - \sum_{i=1}^{M} \lambda_i \upsilon_i = 1 - (1/V) \sum_{i=1}^{M} \lambda_i \theta_i \qquad (2.2)$$

Для уменьшения простоев процессора естественно выбрать его по возможности с меньшим быстродействием. Очевидно, что можно предположить о существовании некоторого оптимального решения, позволяющего определить быстродействие процессора с учетом двух указанных противоречивых факторов. В качестве критерия эффективности при таком подходе может быть использован функционал вида

$$C_V = \beta_0 \eta(V) + \sum_{i=1}^{M} \beta_i \lambda_i w_i(V) \qquad (1.3)$$

где W_i — полное время ожидания заявок в очереди *i*-го типа *(i=1,...,M);* β_i — некоторые весовые, коэффициенты, задаваемые при проектировании ЦУС.

Задание весовых коэффициентов β_i должно осуществляться исходя из назначения системы и требований, предъявляемых к ней. При этом системы, предназначенные для управления объектами или процессами с жесткими требованиями ко времени реакции системы, должны иметь более высокие значения коэффициентов β_i *(i=1, ..., M),* а системы, основное требование к которым—минимум материальных затрат, должны иметь наибольшее значение β_0.

Таким образом, в системах с неограниченным временем пребывания заявок

в качестве критерия эффективности должна быть использована функция (2.1), когда выбирается оптимальная дисциплина обслуживания заявок, и функция (2.3), когда определяется оптимальное значение быстродействия процессора ЦУС.

2.3 Порядок проведения практической работы

2.3.1 Изучить основные теоретические сведения о методах определения эффективности компьютерной системы, разъяснения преподавателя.

2.3.2 Получив вариант задания, рассчитать критерии эффективности компьютерной системы для каждой из трех дисциплин обслуживания.

2.3.3 Исследовать влияние дисциплины обслуживания и статических приоритетов заявок на критерии эффективности компьютерной системы

3 Практическая работа №3. Оценка трудоемкости вычислительного алгоритма

3.1 Общие положения

Трудоемкость алгоритмов – количество вычислительной работы требуемой для реализации алгоритма.

Сложность алгоритма – минимальное количество информации, необходимое для его описания. Обычно в практике сложность алгоритма определяется длиной записи алгоритма в терминах определенной алгоритмической системы. Например, сложность алгоритма можно характеризовать числом операторов в программе или числом команд программы в машинном коде.

Сложность задачи слагается из сложности алгоритма и количества данных. Количество данных, относящихся к задаче, характеризуется числом байтов, посредством которых представляются данные. Располагая сведения о сложности алгоритма и количестве данных, можно определить потребность задачи в ресурсах памяти.

Если сложность алгоритма характеризует потребность алгоритма в памяти, то трудоемкость – его потребность во времени, связанном с периодом работы совокупности устройств, средствами которых реализуется алгоритм. Трудоемкость алгоритма поэтому, иногда называют сложностью вычислений. Оценивается трудоемкость алгоритма количеством операций, выполняемых с целью обработки, ввода и вывода информации в процессе решения задачи. Каждой реализации алгоритма присущ элемент случайности, связанный с тем, что исходные данные представляют собой в общем случае случайную выборку из множества исходных данных, к которым применим алгоритм. Поэтому полная характеристика трудоемкости предполагает описание количества операций, выполняемых за одну реализацию алгоритма, случайными величинами, т.е.

предполагает определение законов распределения числа операций в реализации алгоритма.

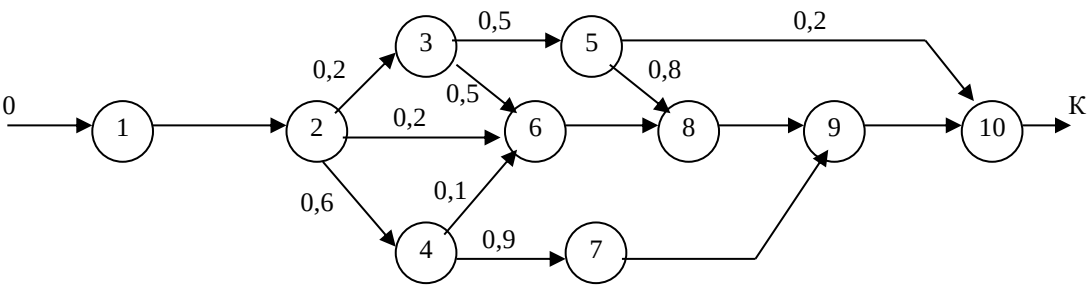
3.2 Сетевой метод оценки трудоемкости алгоритмов

Количество вычислений, производимых при расчетах трудоемкости алгоритмов, можно значительно сократить, если использовать сетевой подход к анализу трудоемкости алгоритмов. Он позволяет получать среднюю, минимальную и максимальную трудоемкость.

Суть сетевого подхода состоит в выделении путей на графе алгоритма, соответствующих минимальной, средней и максимальной трудоемкости последовательности операторов. Эти пути могут быть выделены только на графах, не содержащих циклов. По этой причине методика сетевого подхода начинается с анализа трудоемкости алгоритмов, не содержащих циклы. Затем рассматривается прием исключения циклов из графа алгоритма путем замены их операторами с эквивалентной трудоемкостью. Этот прием позволяет распространить сетевой подход на любые алгоритмы, в том числе содержащие любое количество циклов.

Оценка средней трудоемкости алгоритмов. Алгоритм будем представлять в виде графа, состоящего из K операторных вершин и имеющего единственную конечную вершину с номером $K=K+1;$ дуги графа отмечены вероятностями переходов P_{ij}. Каждой вершине графа поставлено в соответствие среднее значение трудоемкости k_i или l_j. Задача оценки трудоемкости алгоритма сводится к определению среднего числа $n_1,n_2,\ldots,n_k$ обращений к операторам за один прогон алгоритма.

Для применения сетевого подхода к оценке трудоемкости алгоритма, не содержащего циклы, вершины графа алгоритма должны быть пронумерованы в порядке их следования, т.е. так, чтобы любая вершина имела номер, больший любого номера предшествующих ей вершин. Нумерация проводится следующим образом. Начальной вершине присваивается номер 0. Очередной номер $i=1,2,\ldots$ присваивается вершине, в которую входят дуги от уже пронумерованных вершин с номерами, меньшими i. При этом любым двум вершинам должны соответствовать разные номера. Такой порядок нумерации является результативным для любого графа без циклов. Причем конечная вершина графа будет иметь максимальный номер K. Пример корректной нумерации вершин графа:



Поскольку граф не содержит циклов, то при прогоне алгоритма вершина 1 будет выполнена точно один раз, то есть $n_1 = 1$. Среднее число попаданий вычислительного процесса в вершину определяется выражением:

$$n_i = \sum_{j=1}^{k} P_{ji} \cdot n_j \, (i = 2,3,..., k)$$

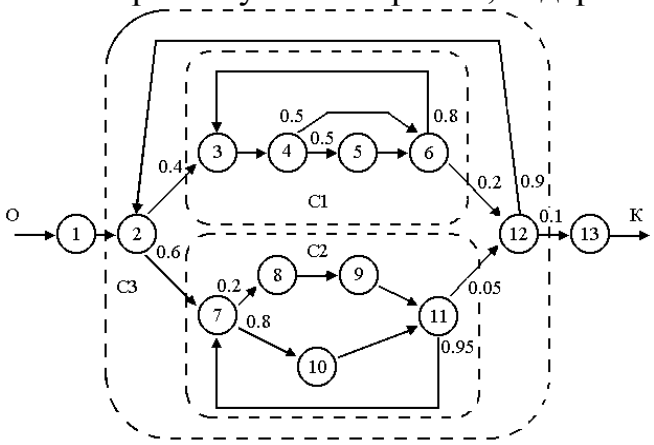
где P_{ji} - вероятность перехода из вершины j в вершину i.

При установленном порядке нумерации вершин на момент вычисления ni значения $n_1, n_2,.,n_{i-1}$ уже определены. Поэтому вычисление значения ni сводится к суммированию произведений, причем поскольку $P_{ij} = 0$ для всех $j \geq 0$, то суммирование следует проводить только для $j<i$.

Определим среднее число обращений $n_1, n_2,.,n_k$ к операторам алгоритма изображенного графа (предыдущий рисунок)

$$n_1 = 1$$
$$n_2 = P_{1,2} \cdot n_1 = 1$$
$$n_3 = P_{2,3} \cdot n_2 = 0.2$$
$$n_4 = P_{2,4} \cdot n_2 = 0.6$$
$$n_5 = P_{3,5} \cdot n_3 = 0.5 \cdot 0.2 = 0.1$$
$$n_6 = P_{3,6} \cdot n_3 + P_{2,5} \cdot n_2 + P_{4,6} \cdot n_4 = 0.36$$
$$n_7 = P_{4,7} \cdot n_4 = 0.9 \cdot 0.6 = 0.54$$
$$n_8 = P_{5,8} \cdot n_5 + P_{6,8} \cdot n_6 = 0.8 \cdot 0.1 + 1 \cdot 0.36 = 0.44$$
$$n_9 = P_{8,9} \cdot n_8 + P_{7,9} \cdot n_7 = 0.98$$
$$n_{10} = P_{5,10} \cdot n_5 + P_{9,10} \cdot n_9 = 0.2 \cdot 0.1 + 1 \cdot 0.98 = 1$$

Рассмотрим случай алгоритма, содержащего циклы:



Непосредственное применение описанной методики к таким алгоритмам невозможно и для вычисления значений $n_1, n_2,.,n_k$ необходимо исключить циклы, заменяя их операторами с эквивалентной трудоемкостью.

Для упрощения описания метода примем, что алгоритм состоит из однотипных операторов, например, только основных операторов. Разделим циклы по рангам. К рангу 1 относятся циклы, которые не содержат внутри себя ни одного цикла, к рангу 2 - циклы, внутри которых есть циклы не выше ранга 1 и так далее. Например, рассматриваемый алгоритм содержит два цикла C1 и C2 ранга 1 и один цикл C2 ранга 2. Совокупность операторов, входящих в цикл, и связывающих их дуг, за исключением дуги, замыкающей цикл, называют телом цикла. Тело цикла ранга 1 является графом без циклов. Применяя к этому графу ранее рассмотренную методику, можно определить значения ni для каждого из операторов, принадлежащих телу цикла, и, следовательно, трудоемкость тела C-го цикла:

$$\sum_{V_j \in C} k_j n_i .$$

Здесь суммирование проводится по всем вершинам V_j, содержащимся в цикле C.

Пусть известно среднее число повторений цикла n_c, равное числу выполнений тела цикла при одном прогоне алгоритма. Если вероятность перехода по дуге, замыкающей цикл, равна Pkl, то n_c равно:

$$\frac{1}{1 - \mathrm{P_{kl}}}$$

В этом случае средняя трудоемкость цикла равна:

$$k_C = n_C \sum_{V_j \in C} k_j n_i$$

и цикл C можно заменить оператором с трудоемкостью k_C.

Применяя указанную процедуру замены циклов операторами ко всем циклам ранга 1, затем к циклам ранга 2 и так далее, в конце концов придем к графу без циклов, трудоемкость которого находится уже рассмотренным способом.

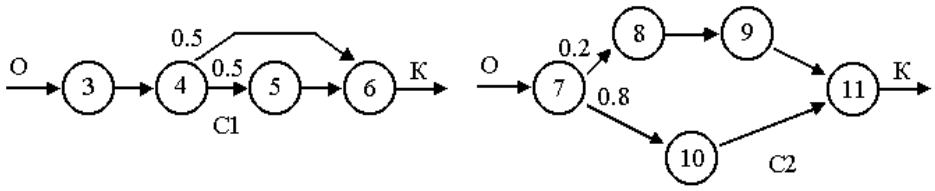
Определим трудоемкость, заданную приведенным выше графом.

Положим, что трудоемкость всех операторов одинакова и равна 1. Среднее число повторений циклов C1, C2 и C3 определяется из вероятностей переходов, указанных на графе, следующими значениями:

$$n_{C1} = \frac{1}{P_{6,12}} = \frac{1}{0.2} = 5$$

$$n_{C2} = \frac{1}{P_{11,12}} = \frac{1}{0.05} = 20$$

$$n_{C3} = \frac{1}{P_{12,13}} = \frac{1}{0.1} = 10$$



Выделим тела циклов С1 и С2 первого ранга:

Применяя к этим графам рассмотренную ранее методику, определяем среднюю трудоемкость k_{C1} и k_{C2} выполнения тел этих циклов:

$$n_7 = 1$$

$$n_3 = 1 \qquad n_8 = P_{7,8} \cdot n_7 = 0.2$$

$$n_4 = P_{3,4} \cdot n_3 = 1 \qquad n_9 = P_{8,5} \cdot n_8 = 0.2$$

$$n_5 = P_{4,5} \cdot n_4 = 0.5 \qquad n_{10} = P_{7,10} \cdot n_7 = 0.8$$

$$n_6 = P_{4,5} \cdot n_4 + P_{5,6} \cdot n_5 = 1 \qquad n_{11} = P_{9,11} \cdot n_9 + P_{10,11} \cdot n_{10} = 1$$

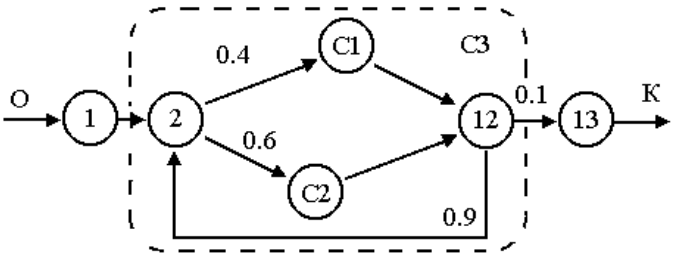
Определяем трудоемкость тел этих циклов:

$$Q_{C1} = \sum_{i=3}^{6} k_i n_i = 3.5 \quad Q_{C2} = \sum_{i=7}^{11} k_i n_i = 3.2$$

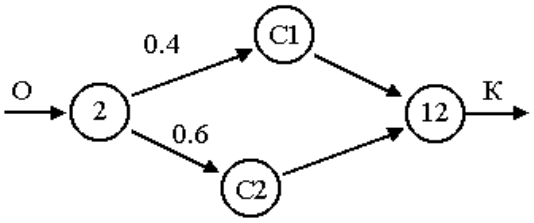
Средняя трудоемкость циклов С1 иС2 вычисляется умножением полученных значений на среднее число повторений этих циклов:

$$\mathrm{k_{C1}} = \mathrm{Q_{C1}} \cdot n_{C1} = 17.5; \quad \mathrm{k_{C2}} = \mathrm{Q_{C2}} \cdot n_{C2} = 64$$

Заменяя в исходном графе циклы С1 и С2 операторами С1 и С2 с трудоемкостью k_{C1} и k_{C2}, получим граф:



Тело цикла С3 имеет вид:



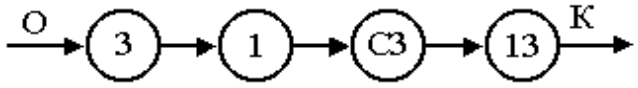
Определим трудоемкость тела цикла С3:

$$n_2 = 1$$
$$n_{C1} = P_{2,C1} \cdot n_2 = 0.4$$
$$n_{C2} = P_{2,C2} \cdot n_2 = 0.6$$
$$n_{12} = P_{C1,12} \cdot n_{C1} + P_{C2,12} \cdot n_{C2} = 1$$
$$Q_{C3} = \sum_{V_i \in C3} k_i n_i = 1 + 17.5 \cdot 0.4 + 64 \cdot 0.6 + 1 = 47.4$$

С учетом числа nС3 повторений цикла трудоемкость цикла составит:

$k_{C3} = Q_{C3} \cdot n_{C3} = 47.4 \cdot 10 = 474$ операций.

Заменив цикл С3 оператором С3 с трудоемкостью k_{C3}, получим граф:



Трудоемкость всего алгоритма, представляемого этим графом, равна:

$\Theta = k_1 + k_{C3} + k_{13} = 1 + 474 + 1 = 476$операций.

3.3 Порядок проведения практической работы

3.3.1 Изучить основные теоретические сведения о методах оценки трудоемкости вычислительных алгоритмов, разъяснения преподавателя.

3.3.2 Получив вариант задания, рассчитать трудоемкость вычислительного алгоритма сетевым методом.

4 Практическая работа №4. Расчет параметров вычислительного конвейера

4.1 Простейшая организация конвейера и оценка его производительности

Разработчики архитектуры компьютеров издавна прибегали к методам проектирования, известным под общим названием "совмещение операций", при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции. Этот общий метод включает два понятия: параллелизм и конвейеризацию. Хотя у них много общего и их зачастую трудно различать на практике, эти термины отражают два совершенно различных подхода. При параллелизме совмещение операций достигается путем воспроизведения в нескольких копиях аппаратной структуры. Высокая производительность достигается за счет одновременной работы всех элементов структур, осуществляющих решение различных частей задачи.

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Так обработку любой машинной команды можно разделить на несколько этапов (несколько ступеней), организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

Для иллюстрации основных принципов построения процессоров мы будем использовать простейшую архитектуру, содержащую 32 целочисленных регистра общего назначения (R0,...,R31), 32 регистра плавающей точки (F0,...,F31) и счетчик команд PC. Будем считать, что набор команд нашего процессора включает типичные арифметические и логические операции, операции с плавающей точкой, операции пересылки данных, операции управления потоком команд и системные операции. В арифметических командах используется трехадресный формат, типичный для RISC-процессоров, а для обращения к памяти используются операции загрузки и записи содержимого регистров в память.

Выполнение типичной команды можно разделить на следующие этапы:

- выборка команды - IF (по адресу, заданному счетчиком команд, из памяти извлекается команда);
- декодирование команды / выборка операндов из регистров - ID;
- выполнение операции / вычисление эффективного адреса памяти - EX;
- обращение к памяти - MEM;
- запоминание результата - WB.

Чтобы конвейеризовать эту схему, можно просто разбить выполнение команд на указанные выше этапы, отведя для выполнения каждого этапа один

такт синхронизации, и начинать в каждом такте выполнение новой команды. Естественно, для хранения промежуточных результатов каждого этапа необходимо использовать регистровые станции. Хотя общее время выполнения одной команды в таком конвейере будет составлять пять тактов, в каждом такте аппаратура будет выполнять в совмещенном режиме пять различных команд.

Работу конвейера можно условно представить в виде временной диаграммы (рисунок 4.1), на которых обычно изображаются выполняемые команды, номера тактов и этапы выполнения команд.

	Номер такта								
Номер команды	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Команда i	IF	ID	EX	MEM	WB				
Команда i+1		IF	ID	EX	MEM	WB			
Команда i+2			IF	ID	EX	MEM	WB		
Команда i+3				IF	ID	EX	MEM	WB	
Команда i+4					IF	ID	EX	MEM	WB

Рисунок 4.1 – Диаграмма работы простейшего конвейера

Конвейеризация увеличивает пропускную способность процессора (количество команд, завершающихся в единицу времени), но она не сокращает время выполнения отдельной команды. В действительности, она даже несколько увеличивает время выполнения каждой команды из-за накладных расходов, связанных с управлением регистровыми станциями. Однако увеличение пропускной способности означает, что программа будет выполняться быстрее по сравнению с простой неконвейерной схемой.

Тот факт, что время выполнения каждой команды в конвейере не уменьшается, накладывает некоторые ограничения на практическую длину конвейера. Кроме ограничений, связанных с задержкой конвейера, имеются также ограничения, возникающие в результате несбалансированности задержки на каждой его ступени и из-за накладных расходов на конвейеризацию. Частота синхронизации не может быть выше, а, следовательно, такт синхронизации не может быть меньше, чем время, необходимое для работы наиболее медленной ступени конвейера. Накладные расходы на организацию конвейера возникают из-за задержки сигналов в конвейерных регистрах (защелках) и из-за перекосов сигналов синхронизации. Конвейерные регистры к длительности такта добавляют время установки и задержку распространения сигналов. В предельном случае длительность такта можно уменьшить до суммы накладных расходов и перекоса сигналов синхронизации, однако при этом в такте не останется времени для выполнения полезной работы по преобразованию информации.

В качестве примера рассмотрим неконвейерную машину с пятью этапами выполнения операций, которые имеют длительность 50, 50, 60, 50 и 50 нс соответственно (рисунок 4.2). Пусть накладные расходы на организацию конвейерной обработки составляют 5 нс. Тогда среднее время выполнения команды в неконвейерной машине будет равно 260 нс. Если же используется конвейерная организация, длительность такта будет равна длительности самого медленного этапа обработки плюс накладные расходы, т.е. 65 нс. Это время соответствует среднему времени выполнения команды в конвейере. Таким образом, ускорение, полученное в результате конвейеризации, будет равно: 260 / 65 = 4.

Конвейеризация эффективна только тогда, когда загрузка конвейера близка к полной, а скорость подачи новых команд и операндов соответствует максимальной производительности конвейера. Если произойдет задержка, то параллельно будет выполняться меньше операций и суммарная производительность снизится.

Рисунок 4.2 – Эффект конвейеризации при выполнении 3-х команд – четырехкратное ускорение

4.2 Порядок проведения практической работы

4.2.1 Изучить основные теоретические сведения о методах конвейеризации вычисление, разъяснения преподавателя.

4.2.2 Получив вариант задания, рассчитать параметры вычислительного конвейера, приведенного на рисунке 4.3, по нижеприведенной методике.

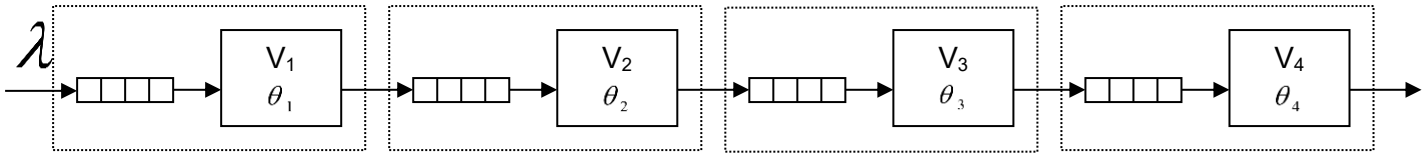


Рисунок 4.3 – Схема конвейера

Исходные данные, ограничения и требования.

Входной поток заявок – одномерный, с интенсивностью λ заявок/сек.;

Операционные блоки (ОБ) конвейера осуществляют бесприоритетное обслуживание.

Количество операционных блоков *N=4.*

Трудоемкость каждого этапа работы конвейера - θ_1, θ_2, θ_3, θ_4.

Быстродействия процессоров операционных блоков V_1, V_2, V_3, V_4.

Коэффициент вариации времени обслуживания заявок $s_i = 1, i = 1,..., N$

1) рассчитать средние времена обслуживания заявок в каждом операционном устройстве (сек.).

Расчетные формулы: Среднее время обслуживания

$$v_i = \theta_i / V_i, \text{ сек}, \tag{4.1}$$

где θ_i – трудоемкость *i*- го этапа работы конвейера, V_i – быстродействие процессора *i*- го операционного блока.

2) рассчитать средние времена ожидания заявок в каждом операционном устройстве (сек.) $w_i, i = 1...N$

Расчетные формулы: Для бесприоритетной дисциплины обслуживания среднее время ожидания заявок в *i*- м операционном устройстве определяется выражением (см. практ. занятие №1)

$$w_i = \frac{\rho_i v_i (1+s_i^2)}{2(1-\rho_i)} = \frac{\rho_i v_i (1+1^2)}{2(1-\rho_i)} = \frac{2\rho_i v_i}{2(1-\rho_i)} = \frac{\rho_i v_i}{1-\rho_i}, \text{ сек}, \tag{4.2}$$

где ρ_i - суммарная загрузка *i*- го операционного блока и $\rho_i = \lambda_i v_i$.

3) рассчитать средние длины очередей в каждом операционном устройстве (шт.) $l_i, i = 1...N$

Расчетные формулы: Средняя длина очереди в *i*- м операционном устройстве $l_i = \lambda_i w_i$

4) приняв размер заявки равным *S=2048* байтов, рассчитать размеры буферов каждого операционного устройства с запасом размера буфера в 50% (байтов) $L_i, i = 1...N$

Расчетные формулы: Размер буфера в *i*- м операционном устройстве

$$L_i = l_i S(1+K)\text{, байтов,} \quad (4.3)$$

где K=0,5 – коэффициент запаса размера буфера в 50%.

5) рассчитать средние времена пребывания заявок в каждом операционном устройстве $u_i, i = 1...N$, сек.

Расчетные формулы:

$$u_i = w_i + v_i \quad (4.4)$$

6) рассчитать среднее время загрузки конвейера $T_{загр}$, сек

Расчетные формулы:

$$T_{загр} = \sum_{i=1}^{N} u_i \quad (4.5)$$

7) рассчитать средний такт конвейера $T_{конв}$, сек.

Расчетные формулы:

$$T_{конв} = MAX[v_i, i = 1...N] \quad (4.6)$$

8) рассчитать среднюю системную производительность конвейера $\lambda_{конв}$, задач/сек.

Расчетные формулы:

$$\lambda_{конв} = 1/T_{конв} \quad (4.7)$$

9) результаты расчетов свести в таблицу

Параметры	*vi, сек*	w_i, *сек*	l_i, *шт*	L_i, *байтов*	u_i, *сек*	*Тзагр, сек*	*Тконв, сек*	$\lambda_{конв}$, *задач/сек*
ОБ 1								
ОБ 2								
ОБ 3								
ОБ 4								

10) рассчитать коэффициент увеличения производительности за счет конвейеризации (коэффициент ускорения).

Расчетные формулы:

$$P_{конв} = \lambda_{конв} / \lambda_{послед}\text{,} \quad (4.8)$$

где $\lambda_{послед}$ - производительность последовательной обработки (без использования конвейера).

Для расчета $\lambda_{послед}$ выполнить следующие операции:

а) общая трудоемкость θ выполнения средней задачи очевидно равна сумме трудоемкостей выполнения всех этапов конвейера:

$$\theta = \sum_{i=1}^{N} \theta_i \ . \tag{4.9}$$

б) пусть быстродействие последовательного устройства будет равно максимальному из быстродействий операционных устройств конвейера:

$$\mathrm{V}_{\text{послед}} = \mathrm{MAX}[V_i, i = 1...N], \tag{4.10}$$

в) тогда среднее время обслуживания задачи в последовательном устройстве

$$v_{послед} = \theta / V_{послед}, \ сек. \tag{4.11}$$

е) тогда

$$\lambda_{\text{послед}} = 1/V_{\text{послед}} \text{, задач/сек.} \tag{4.12}$$

ж) выполняем расчет по формуле (4.8).

10) сделать выводы по влиянию конвейеризации на производительность обработки задач

5 Практическая работа №5. Алгоритмы маршрутизации сетевой информации

5.1 Общие положения

Пакеты поступают в сеть передачи данных (СПД), имея в своем заголовке адрес порта назначения. Узел связи СПД, в который поступил пакет, должен по адресу порта назначения определить *маршрут* передачи пакета—выходную линию связи, в которую нужно передать пакет. При передаче данных по виртуальному каналу маршрутизация выполняется единственный раз, когда устанавливается виртуальное соединение. При передаче данных в форме дейтаграмм маршрутизация выполняется для каждого отдельного пакета.

Выбор маршрутов в узлах связи СПД производится по *алгоритму маршрутизации*—правилу назначения выходной линии связи на основе данных, содержащихся в заголовке пакета, и данных, представляющих состояние узла связи и, возможно, СПД в целом. Эффективность алгоритма маршрутизации характеризуется следующими показателями: 1) временем доставки пакетов; 2) нагрузкой, создаваемой на сеть потоками пакетов, поступающими в сеть и распределяемыми по линиям и узлам связи; 3) затратами ресурсов в узлах связи, в первую очередь — затратами памяти и времени процессора коммутационной

ЭВМ. Первые два показателя — основные при оценке эффективности. Алгоритмы маршрутизации имеют целью обеспечить непрерывное продвижение пакетов от источников к адресатам. При этом алгоритм стремится выбрать наиболее подходящее направление передачи пакета — с минимальным временем доставки или наиболее полным использованием пропускной способности СПД. Эффективность алгоритмов маршрутизации уменьшается в связи со следующими факторами: 1) передачей пакета в направлении, не приводящем к минимальному времени доставки; 2) передачей пакета в узел связи, находящийся под высокой нагрузкой; 3) созданием дополнительной нагрузки на сеть за счет передачи служебной информации, необходимой для выполнения алгоритма маршрутизации.

Задача маршрутизации решается в следующих условиях. Сеть передачи данных имеет произвольную ячеистую структуру. Кратчайший маршрут, обеспечивающий доставку пакета за минимальное время, зависит от двух основных факторов:

1) топологии СПД и пропускной способности линий связи;

2) нагрузки на линии связи, определяемой числом пакетов, стоящих в очереди на передачу в каждом узле связи.

Топология сети изменяется в результате отказов линий и узлов связи и отчасти при развитии СПД—введении новых линий и узлов связи. Пропускная способность линий связи зависит от уровня помех и параметров аппаратуры, обслуживающей линии. Нагрузка на линии связи—наиболее динамичный фактор, изменяющийся крайне быстро и в труднопрогнозируемом направлении. Следовательно, чтобы решение задачи было оптимальным, необходимо каждому узлу представлять информацию о состоянии СПД в целом — всех узлов и линий связи. Информация о текущей топологии сети и пропускной способности линий может быть представлена узлам. Однако не существует способа точно предсказать состояние нагрузки в сети. Поэтому алгоритмы маршрутизации могут использовать данные о состоянии нагрузки, запаздывающие по отношению к текущему моменту времени из-за конечной скорости передачи данных и не соответствующие последующим моментам времени, в которые будет принято решение о направлении передачи пакетов. Таким образом, во всех случаях алгоритмы маршрутизации работают в условиях неопределенности текущего и будущего состояния СПД и пульсирующей нагрузки, создаваемой абонентами.

5.2 Классификация алгоритмов маршрутизации

Классификация алгоритмов маршрутизации производится в зависимости от направленности передачи пакетов и способов представления данных о топологии и нагрузке сети.

Простая маршрутизация — способ маршрутизации, не изменяющийся при изменении топологии и состояния СПД. Простая маршрутизация обеспечивается разными алгоритмами, типичными из которых являются алгоритмы случайной и лавинной маршрутизации.

Случайная маршрутизация — передача пакета из узла в любом, случайным образом выбранном направлении, кроме направления, по которому пакет поступил в узел. Пакет, совершая «блуждания» по сети, с конечной вероятностью когда-либо достигает адресата. Очевидно, что случайная маршрутизация неэффективна ни по времени доставки пакетов, ни по использованию пропускной способности сети.

Лавинная маршрутизация — передача пакета из узла во всех направлениях, кроме того, по которому поступил пакет. При этом, если узел связан с *n* Другими узлами СПД, пакет передается в *n*—1 направлениях, т. е. размножается. Очевидно, что хотя бы одно направление обеспечит доставку пакета за минимальное время, т. е. лавинная маршрутизация гарантирует малое время доставки, однако это достигается за счет резкого ухудшения использования пропускной способности сети из-за загрузки ее большим числом пакетов.

Маршрутизация по предыдущему опыту — передача пакета в направлении, выбираемом на основе анализа потока, проходящего через узел. При этом пакеты, поступая в сеть, снабжаются адресами получателя и источника и счетчиком числа пройденных узлов. Пакет, который пришел в узел со значением счетчика 1, определяет соседний узел; пакет со значением счетчика 2 определяет узел, находящийся на расстоянии двух шагов, и т. д. Эти данные, позволяют установить топологию сети и на ее основе построить таблицу для выбора маршрута. Постоянно анализируя ' число пройденных узлов, можно изменять таблицу маршрутов, если появился пакет с числом пройденных узлов, меньшим ранее зарегистрированного. Этот способ маршрутизации позволяет узлам приспосабливаться к изменению топологии сети, однако процесс адаптации протекает медленно и неэффективно. Метод изучения пути передачи пакетов используется для построения ряда модификаций алгоритмов простой маршрутизации.

Простая маршрутизация, не обеспечивая направленной передачи пакетов от источников к адресатам, имеет низкую эффективность. Основное ее достоинство—обеспечение устойчивой работы СПД при выходе из строя различных частей сети.

Фиксированная маршрутизация — способ выбора направления передачи по таблице маршрутизации, устанавливающей направление передачи для каждого узла назначения. Таблицы маршрутизации определяют кратчайшие пути от узлов к адресатам и вводятся в узлы связи, например, от управляющего центра сети. Для слабозагруженных сетей этот способ маршрутизации дает хорошие результаты, но его эффективность падает по мере увеличения нагрузки на сеть. При отказе линий связи необходимо менять таблицу маршрутизации. Для этого можно, например, размещать в каждом узле связи набор таблиц маршрутизации, подготовленных на случай отказа одной из линий связи. При возникновении отказа по узлам сети рассылается управляющий пакет, содержащий сведения об отказе, реагируя на который, узлы меняют таблицы маршрутизации путем выбора соответствующих таблиц из хранимого набора. Очевидно, что разработать способ фиксированной маршрутизации, обеспечивающей работоспособность сети при отказе многих линий, является чрезвычайно

трудной задачей. К тому же фиксированная маршрутизация не позволяет адаптироваться к изменениям нагрузки, что в общем случае приводит к значительным задержкам пакетов в СПД. Фиксированная маршрутизация может строиться на основе единственного пути передачи пакетов между двумя абонентами. Такой способ называется *однопутевой маршрутизацией.* Его недостаток — неустойчивость к отказам и перегрузкам. Для повышения устойчивости в таблицах маршрутизации указывается несколько возможных путей передачи пакета и вводится правило выбора целесообразного пути. Такой способ называется *многопутевой маршрутизацией.*

Адаптивная маршрутизация — способ выбора направления передачи, учитывающий изменение состояния СПД. При адаптивной маршрутизации узлы СПД принимают решение о выборе маршрутов, реагируя на разного рода данные об изменении топологии и нагрузки. В идеальном случае каждый узел сети для принятия решения должен располагать полной информацией о текущем состоянии всех остальных узлов, о топологии сети и длине очередей к каждому направлению в каждом узле. Однако, как показали исследования, даже в этом идеальном случае задержки в СПД лишь немногим меньше, чем при фиксированной маршрутизации, таблицы которой определяют кратчайшие пути в сети и не изменяются при колебаниях нагрузки. Дело в том, что оптимальные маршруты, формируемые на основе самой свежей информации о распределении нагрузки в сети, становятся неоптимальными в последующие моменты времени, когда пакеты еще не достигли адресатов. Когда, например, сильнозагруженные узлы получают информацию о том, что некоторая часть сети загружена слабо, они одновременно направляют пакеты в эту часть сети, создавая в сети, быть может, худшую ситуацию, чем предшествующая. Таким образом, алгоритмы адаптивной маршрутизации не обеспечивают оптимальности маршрутов. Однако выбор даже не оптимального, а близкого к нему маршрута приводит к значительному уменьшению времени доставки, особенно при пиковых нагрузках, а также к некоторому увеличению пропускной способности сети. Поэтому адаптивная маршрутизация получила широкое применение в вычислительных сетях и в первую очередь—в сетях с большим числом узлов связи (10 и более).

Алгоритмы адаптивной маршрутизации классифицируются по информации, используемой ими для принятия решений при назначении маршрутов.

Локальная адаптивная маршрутизация основана на использовании информации, имеющейся в отдельном узле СПД. Эта информация включает в себя: 1) таблицу маршрутизации, определяющую все направления передачи пакетов; 2) данные о текущем состоянии выходных каналов (работают или не работают); 3) длину очередей пакетов, ожидающих передачи по выходным каналам. Информация о состояниях других узлов сети не используется. Таблицы маршрутизации указывают кратчайшие маршруты, проходящие через минимальное число узлов и обеспечивающие передачу пакета в узел назначения за минимальное время.

Распределенная адаптивная маршрутизация основана на использовании информации, получаемой от соседних узлов сети. Этот способ маршрутизации может реализоваться, например,

следующим образом. Каждый узел сети формирует таблицы маршрутов ко всем узлам назначения, минимизирующие задержки в сети, причем для каждого маршрута указывается фактическое время передачи пакета в узел назначения. До начала работы сети это время оценивается исходя из топологии сети. В процессе работы сети узлы регулярно обмениваются с соседними узлами таблицами задержки. После обмена каждый узел пересчитывает задержки с учетом поступивших данных и длины очередей в самом узле. Полученные значения используются для выбора маршрутов: пакет ставится в очередь к маршруту, который характеризуется минимальным временем доставки. Обмен таблицами задержки производится периодически или когда обнаруживаются существенные изменения задержки из-за изменения очередей на передачу или состояния линий связи в результате отказа. Естественно, что периодический обмен таблицами задержки значительно увеличивает загрузку сети, а асинхронный — снижает. Однако в каждом случае загрузка остается весьма существенной и к тому же сведения об изменении состояния узлов медленно распространяются по сети. Так, при обмене с интервалом 2/3 с время передачи данных составляет несколько секунд, и в этот период узлы направляют пакеты по старым путям, что может создать перегрузку в районе вышедших из строя компонентов сети.

Централизованная адаптивная маршрутизация основана на использовании информации, получаемой от центра маршрутизации. При этом каждый узел сети формирует сообщения о своем состоянии—длине очередей, работоспособности линий связи и эти сообщения передаются в центр маршрутизации. Последний на основе полученных данных формирует таблицы маршрутизации, рассылаемые всем узлам сети. Неизбежные временные задержки при передаче данных в центр маршрутизации, формировании и рассылке таблиц приводят к потере эффективности централизованной маршрутизации, особенно в ситуациях, когда нагрузка сильно пульсирует. Поэтому централизованная маршрутизация по эффективности не превосходит локальную адаптивную, а кроме того, отличается специфичным недостатком — потерей управления сетью при отказе центра маршрутизации.

Гибридная адаптивная маршрутизация основана на использовании таблиц, периодически рассылаемых центром маршрутизации, в сочетании с анализом длины очередей в узлах. Если таблица маршрутизации, сформированная для узла связи центром, определяет единственное направление передачи пакета, то пакет передается именно в этом направлении. Если же таблица определяет несколько направлений, то узел выбирает направление в зависимости от текущих значений длины очередей — по алгоритму локальной адаптивной маршрутизации. Гибридная маршрутизация компенсирует недостатки централизованной и локальной: маршруты, формируемые центром, являются устаревшими, но соответствуют глобальному состоянию сети; локальные алгоритмы являются «близорукими», но обеспечивают своевременность решений.

5.3 Протокол маршрутной информации RIP

Протокол маршрутной информации RIP (Routing Information Protocol) — один из самых простых протоколов динамической маршрутизации. Применяется в небольших компьютерных сетях, позволяет маршрутизаторам динамически обновлять маршрутную информацию (направление и дальность в хопах), получая ее от соседних маршрутизаторов.

Алгоритм маршрутизации RIP (алгоритм Беллмана — Форда) был впервые разработан в 1969 году, как основной для сети ARPANET.

RIP — так называемый протокол дистанционно-векторной маршрутизации, который оперирует транзитными участками (хоп, hop) в качестве метрики маршрутизации. Максимальное количество транзитных участков, разрешенное в RIP — 15 (метрика 16 означает «бесконечно большую метрику»). Каждый RIP-маршрутизатор по умолчанию вещает в сеть свою полную таблицу маршрутизации раз в 30 секунд, довольно сильно нагружая низкоскоростные линии связи. RIP работает в сетях TCP/IP, используя UDP порт 520.

В современных сетевых средах RIP — не самое лучшее решение для выбора в качестве протокола маршрутизации, так как его возможности уступают более современным протоколам, таким как EIGRP, OSPF. Ограничение на 15 транзитных участков не дает применять его в больших сетях. Преимущество этого протокола — простота конфигурирования.

5.4 Протокол OSPF

OSPF (Open Shortest Path First) — протокол динамической маршрутизации, основанный на технологии отслеживания состояния канала (link-state technology) и использующий для нахождения кратчайшего пути *алгоритм Дейкстры*.

Протокол OSPF был разработан IETF в 1988 году. Последняя версия протокола представлена в RFC 2328. Протокол OSPF представляет собой протокол внутреннего шлюза (Interior Gateway Protocol — IGP). Протокол OSPF распространяет информацию о доступных маршрутах между маршрутизаторами одной автономной системы.

OSPF имеет следующие преимущества:

- высокая скорость сходимости по сравнению с дистанционно-векторными протоколами маршрутизации;
- поддержка сетевых масок переменной длины (VLSM, см. п.9.2.3);
- оптимальное использование пропускной способности с построением дерева кратчайших путей.

Принцип работы заключается в следующем:

1. После включения маршрутизаторов протокол ищет непосредственно подключенных соседей и устанавливает с ними «дружеские» отношения.

2. Затем они обмениваются друг с другом информацией о подключенных и доступных им сетях. То есть они строят карту сети (топологию сети). Данная карта одинакова на всех маршрутизаторах.

3. На основе полученной информации запускается алгоритм SPF (Shortest Path First, «выбор наилучшего пути»), который рассчитывает оптимальный маршрут к каждой сети. Данный процесс похож на построение дерева, корнем которого является сам маршрутизатор, а ветвями — пути к доступным сетям. Данный процесс, то есть конвергенция, происходит очень быстро.

Алгоритм Дейкстры — алгоритм на графах, изобретённый нидерландским учёным Эдсгером Дейкстрой в 1959 году. Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.

Формальное определение: дан взвешенный ориентированный граф *G(V, E)* без дуг отрицательного веса. Найти кратчайшие пути от некоторой вершины *a* графа *G* до всех остальных вершин этого графа.

Каждой вершине из *V* сопоставим метку — минимальное известное расстояние от этой вершины до *a*.

Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки.

Работа алгоритма завершается, когда все вершины посещены.

Инициализация. Метка самой вершины *a* полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от *a* до других вершин пока неизвестны.

Все вершины графа помечаются как непосещённые.

Шаг алгоритма. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина *u*, имеющая минимальную метку.

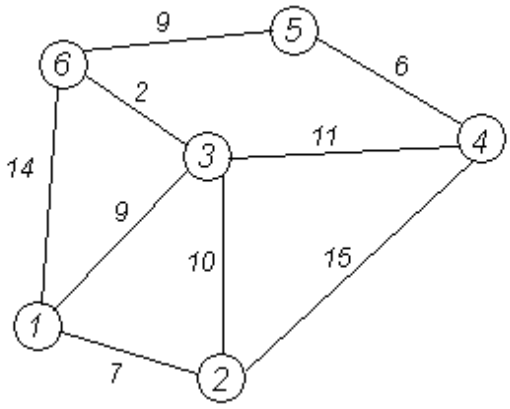
Мы рассматриваем всевозможные маршруты, в которых *u* является предпоследним пунктом. Вершины, в которые ведут рёбра из u, назовём соседями этой вершины. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки *u* и длины ребра, соединяющего *u* с этим соседом.

Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину *u* как посещённую и повторим шаг алгоритма.

Пример

Рассмотрим выполнение алгоритма на примере графа, показанного на рисунке ниже.

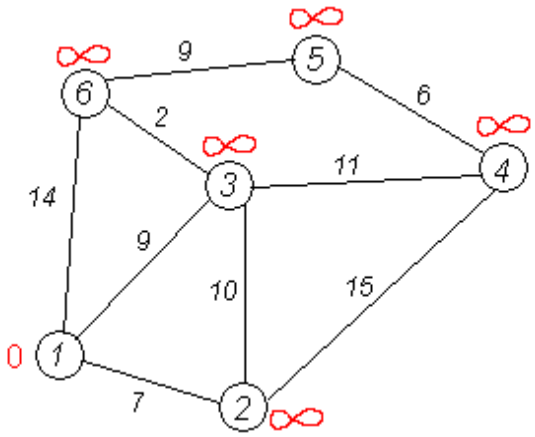
Пусть требуется найти кратчайшие расстояния от 1-й вершины до всех остальных.



Кружками обозначены вершины, линиями — пути между ними (рёбра графа).

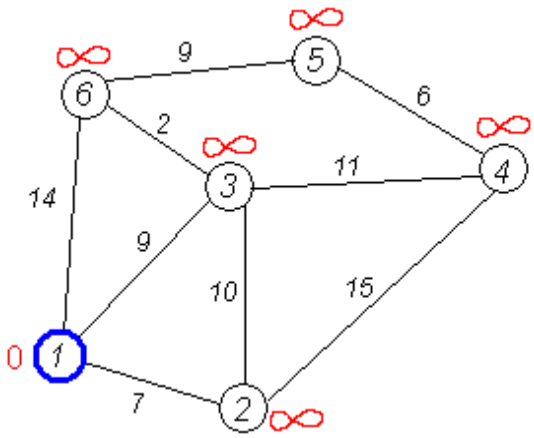
В кружках обозначены номера вершин, над рёбрами обозначен их вес — длина пути.

Рядом с каждой вершиной красным обозначена метка — длина кратчайшего пути в эту вершину из вершины 1.



Первый шаг.

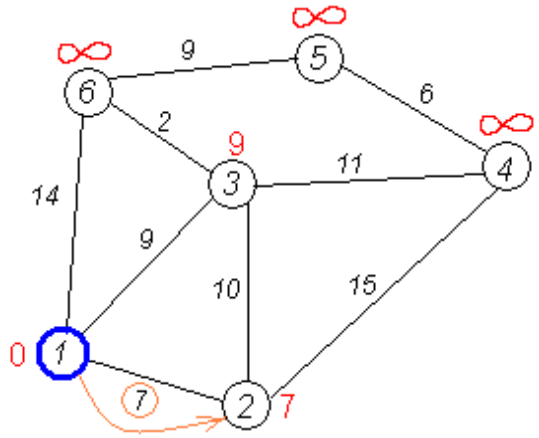
Минимальную метку имеет вершина 1. Её соседями являются вершины 2, 3 и 6.



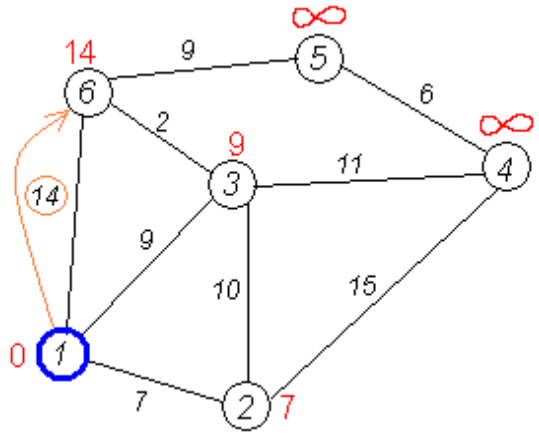
Первый по очереди сосед вершины 1 — вершина 2, потому что длина пути до неё минимальна.

Длина пути в неё через вершину 1 равна сумме значения метки вершины 1 и длины ребра, идущего из 1-й в 2-ю, то есть 0 + 7 = 7.

Это меньше текущей метки вершины 2, бесконечности, поэтому новая метка 2-й вершины равна 7.



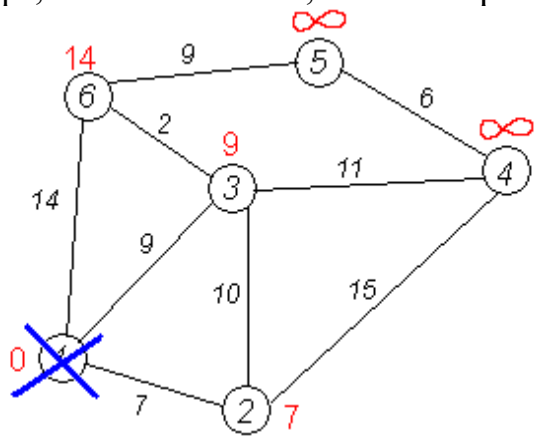
Аналогичную операцию проделываем с двумя другими соседями 1-й вершины — 3-й и 6-й.



Все соседи вершины 1 проверены.

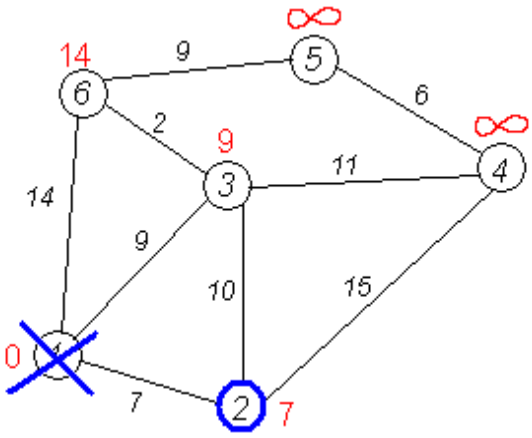
Текущее минимальное расстояние до вершины 1 считается окончательным и пересмотру не подлежит.

Вычеркнем её из графа, чтобы отметить, что эта вершина посещена.



Второй шаг.

Снова находим «ближайшую» из непосещённых вершин. Это вершина 2 с меткой 7.

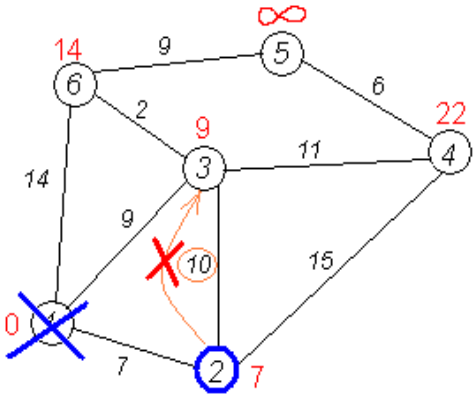


Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через 2-ю вершину. Соседями вершины 2 являются вершины 1, 3 и 4.

Первый (по порядку) сосед вершины 2 — вершина 1. Но она уже посещена, поэтому с 1-й вершиной ничего не делаем.

Следующий сосед — вершина 3, так как имеет минимальную метку.

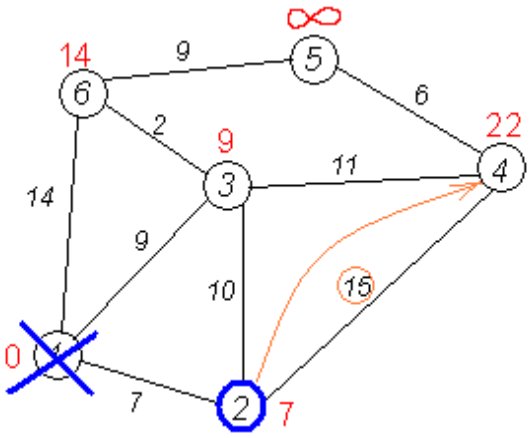
Если идти в неё через 2, то длина такого пути будет равна 17 (7 + 10 = 17). Но текущая метка третьей вершины равна 9, а это меньше 17, поэтому метка не меняется.



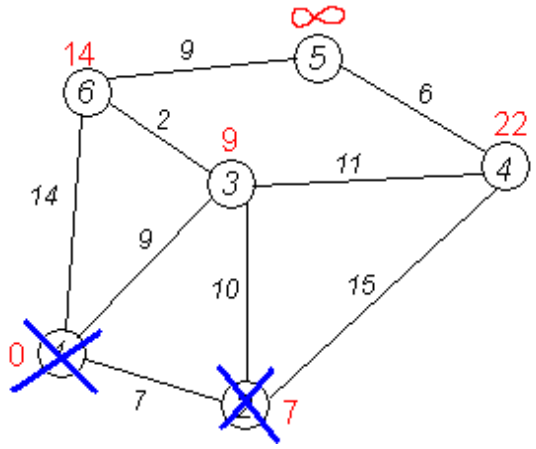
Ещё один сосед вершины 2 — вершина 4.

Если идти в неё через 2-ю, то длина такого пути будет равна сумме кратчайшего расстояния до 2-й вершины и расстояния между вершинами 2 и 4, то есть 22 (7 + 15 = 22).

Поскольку 22<∞, устанавливаем метку вершины 4 равной 22.

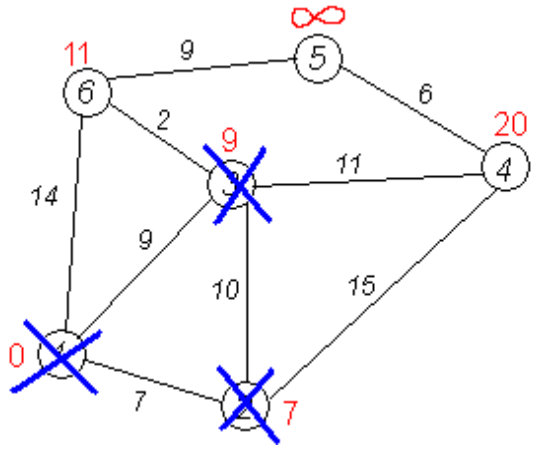


Все соседи вершины 2 просмотрены, замораживаем расстояние до неё и помечаем её как посещённую.



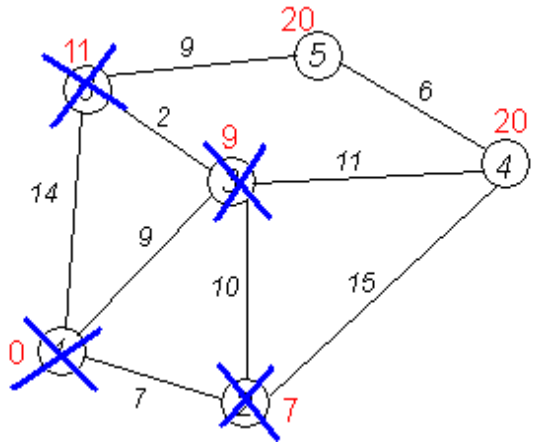
Третий шаг.

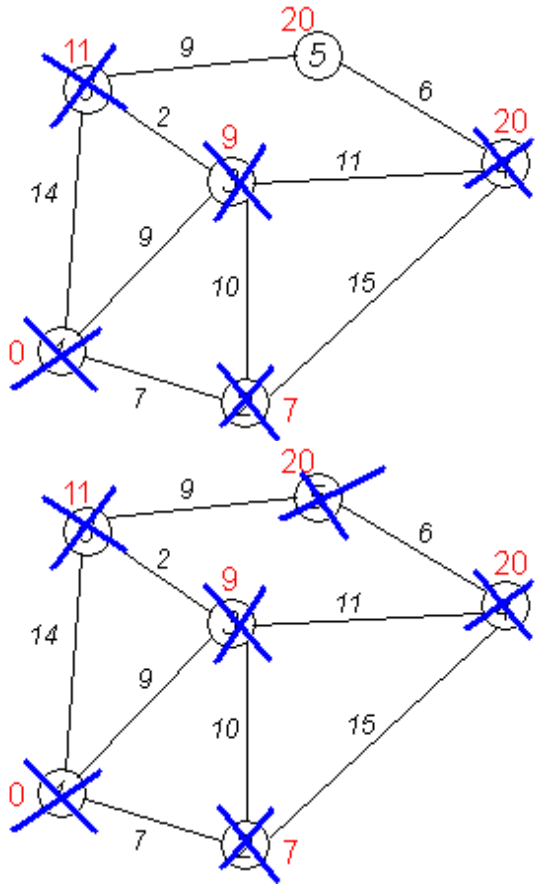
Повторяем шаг алгоритма, выбрав вершину 3. После её «обработки» получим такие результаты:



Дальнейшие шаги.

Повторяем шаг алгоритма для оставшихся вершин. Это будут вершины 6, 4 и 5, соответственно порядку.





Завершение выполнения алгоритма.

Алгоритм заканчивает работу, когда все вершины посещены.

Результат работы алгоритма виден на последнем рисунке: кратчайший путь от вершины 1 до 2-й составляет 7, до 3-й — 9, до 4-й — 20, до 5-й — 20, до 6-й — 11.

Если в какой-то момент все непосещённые вершины помечены бесконечностью, то это значит, что до этих вершин нельзя добраться (то есть граф несвязный). Тогда алгоритм может быть завершён досрочно.

5.5 Порядок проведения практической работы

5.5.1 Изучить основные теоретические сведения об алгоритмах маршрутизации сетевой информации, разъяснения преподавателя.

5.5.2 Получив вариант задания, рассчитать кратчайшие маршруты от указанных узлов маршрутизации.

6 Практическая работа №6. IP-адресация

Задача – приобрести навыки: определения адреса подсети и адреса хоста по маске подсети; определения количества и диапазона адресов возможных узлов в подсетях.

6.1 Задание 1

Определить, находятся ли два узла A и B в одной подсети или в разных подсетях, если адреса компьютера A и компьютера B соответственно равны: 26.219.123.6/10 и 26.218.102.31, маска подсети 255.192.0.0.

Указания к выполнению

1. Переведите адреса компьютеров и маску в двоичный вид.
2. Для получения двоичного представления номеров подсетей обоих узлов выполните операцию логического умножения AND над IP-адресом и маской каждого компьютера.
3. Двоичный результат переведите в десятичный вид.
4. Сделайте вывод.

Процесс решения можно записать следующим образом:

Компьютер A:

IP-адрес: 26.219.123.6 = 00011010 11011011 01111011 00000110
Маска подсети: 255.192.0.0 = 11111111 11000000 00000000 00000000

Компьютер B:

IP-адрес: 26.218.102.31 = 00011010 11011010 01100110 00011111
Маска подсети: 255.192.0.0 = 11111111 11000000 00000000 00000000

Получаем номер подсети, выполняя операцию AND над IP-адресом и маской подсети.

Компьютер A:

```
        00011010 11011011 01111011 00000110
AND     11111111 11000000 00000000 00000000
-------------------------------------------
        00011010 11000000 00000000 00000000
           26.      192.      0.        0
```

Компьютер B:

```
        00011010 11011010 01100110 00011111
AND     11111111 11000000 00000000 00000000
-------------------------------------------
        00011010 11000000 00000000 00000000
           26.      192.      0.        0
```

Ответ: номера подсетей двух IP-адресов совпадают, значит компьютеры A и B находятся в одной подсети. Следовательно, между ними возможно установить прямое соединение без применения шлюзов.

6.2 Задание 2

Определить количество и диапазон IP-адресов в подсети, если известны номер подсети и маска подсети.

Номер подсети – 26.219.128.0, маска подсети – 255.255.192.0.

Указания к выполнению

1. Переведите номер и маску подсети в двоичный вид.

Номер подсети: 26.219.128.0 = 00011010 11011011 10000000 00000000
Маска подсети: 255.255.192.0 = 11111111 11111111 11000000 00000000

2. По маске определите количество бит, предназначенных для адресации узлов (их значение равно нулю). Обозначим их буквой К.

3. Общее количество адресов равно 2^K. Но из этого числа следует исключить комбинации, состоящие из всех нулей или всех единиц, так как данные адреса являются особыми. Следовательно, общее количество узлов подсети будет равно $2^K – 2$.

В рассматриваемом примере K = 14, $2^K – 2$ = 16 382 адресов.

4. Чтобы найти диапазон IP-адресов нужно найти начальный и конечный IP-адреса подсети. Для этого выделите в номере подсети те биты, которые в маске подсети равны единице. Это разряды, отвечающие за номер подсети. Они будут совпадать для всех узлов данной подсети, включая начальный и конечный:

Номер подсети: 26.219.128.0 = **00011010 11011011 10**000000 00000000
Маска подсети: 255.255.192.0 = **11111111 11111111 11**000000 00000000

5. Чтобы получить начальный IP-адрес подсети нужно невыделенные биты в номере подсети заполнить *нулями*, за исключением крайнего правого бита, который должен быть равен единице. Полученный адрес будет первым из допустимых адресов данной подсети:

Начальный адрес: 26.219.128.1 = **00011010 11011011 10**000000 00000001
Маска подсети: 255.255.192.0 = **11111111 11111111 11**000000 00000000

6. Чтобы получить конечный IP-адрес подсети нужно невыделенные биты в номере подсети заполнить *единицами*, за исключением крайнего правого бита, который должен быть равен нулю. Полученный адрес будет последним из допустимых адресов данной подсети:

Конечный адрес: 26.219.191.254 = **00011010 11011011 10**111111 11111110
Маска подсети: 255.255.192.0 = **11111111 11111111 11**000000 00000000

Ответ: Для подсети 26.219.128.0 с маской 255.255.192.0:
количество возможных адресов: 16 382,
диапазон возможных адресов: 26.219.128.1 – 26.219.191.254.

6.3 Порядок проведения практической работы

6.3.1 Изучить основные теоретические сведения об IP-адресации в компьютерных сетях, ход решения задания 1, задания 2, разъяснения преподавателя.

63.2 Получив вариант заданий, решить поставленные задачи. Выполненные задания оформляются также (с той же степенью детализации), как и в вышеприведенных примерах.

7 Практическая работа №7. Структурирование IP-сети с использованием масок

Задача – приобрести навыки структурирования сети с использованием масок.

7.1 Задание

Организации выделена сеть класса C: 212.100.54.0/24. Требуется разделить данную сеть на N=4 подсети с количеством узлов в каждой не менее M=50. Определить маски, количество и диапазон возможных адресов новых подсетей.

Указания к выполнению

1. В сетях класса C (маска содержит 24 единицы – 255.255.255.0) под номер узла отводится 8 бит, т. е. сеть может включать $2^8 – 2 = 254$ узла.
2. Требование деления на 4 подсети по 50 узлов в каждой может быть выполнено: $4 \cdot 50 = 200 < 254$. Однако число узлов в подсети должно быть кратно степени двойки. Относительно 50 ближайшая б*о*льшая степень – $2^6 = 64$. Следовательно, для номера узла нужно отвести 6 бит, вместо 8, а маску расширить на 2 бита – до 26 бит (рисунок 7.1).
3. В этом случае вместо одной сети с маской 255.255.255.0 образуется 4 подсети с маской 255.255.255.192 и количеством возможных адресов в каждой – $2^6-2=62$ (не забывайте про два особых адреса).
4. Номера новых подсетей отличаются друг от друга значениями двух битов, отведенных под номер подсети. Эти биты равны 00, 01, 10, 11.

Ответ:

Маска подсети – 255.255.255.192 (/26)

Количество возможных адресов в каждой из подсетей – 62.

Адрес подсети 1: 212.100.54.0

Диапазон адресов узлов в подсети 1: 212.100.54.1 – 212.100.54.62.

Адрес подсети 2: 212.100.54.64

Диапазон адресов узлов в подсети 2: 212.100.54.65 – 212.100.54.126.

Адрес подсети 3: 212.100.54.128

Диапазон адресов узлов в подсети 3: 212.100.54.129 – 212.100.54.190.

Адрес подсети 4: 212.100.54.192

Диапазон адресов узлов в подсети 4: 212.100.54.193 – 212.100.54.254.

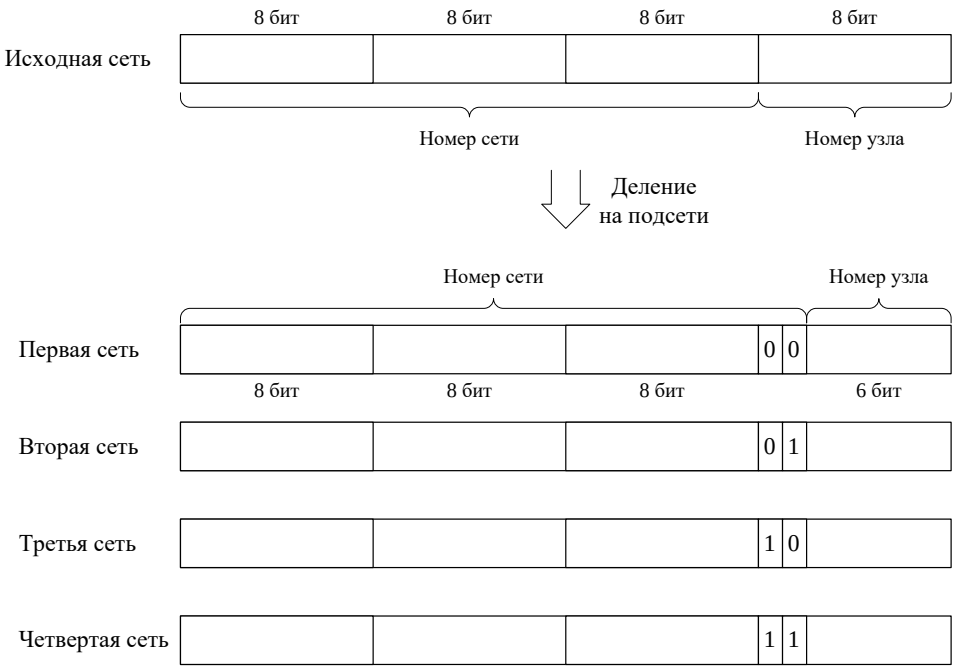


Рисунок 7.1 – Адреса подсетей после деления

На рисунке 7.2 приведена схема полученных подсетей.

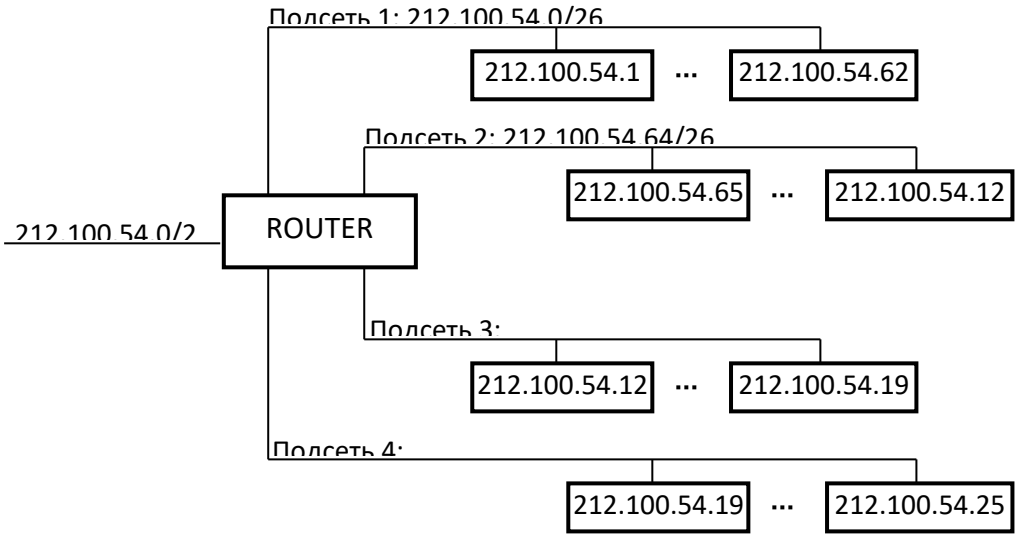


Рисунок 7.2 – Схема полученных подсетей. Примечание: для каждой из подсетей указаны младший и старший IP-адреса конечных узлов

7.2 Порядок проведения практической работы

7.2.1 Изучить основные теоретические сведения об IP-адресации в компьютерных сетях, методах структурирования сети с использованием масок, ход решения задания, разъяснения преподавателя.

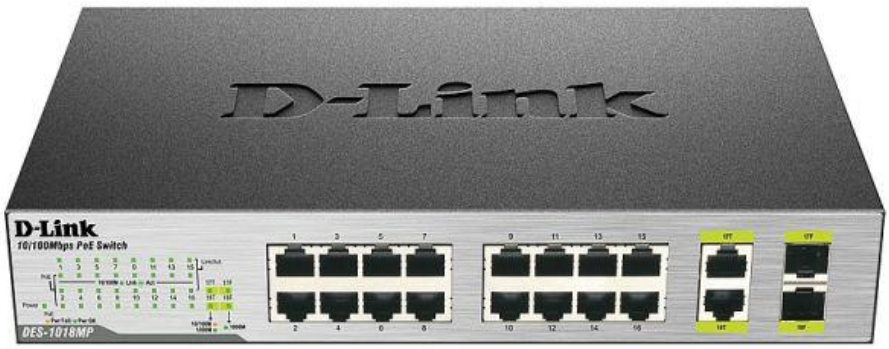
7.2.2 Получив вариант заданий, выполнить вышеописанное задание. Выполненные задания оформляются также (с той же степенью детализации), как и в вышеприведенных примерах.

Примечание: данное задание базируется на навыках, приобретенных в результате выполнения предыдущей практической работы №6.

8 Практическое занятие №8. Изучение характеристик коммутаторов локальных вычислительных сетей

8.1 Пример выполнения расчетов

Объект исследования – коммутатор DES-1018MP (www.d-link.ru). Неуправляемый коммутатор с 16 портами 10/100Base-TX, 2 комбо-портами 100/1000Base-T/SFP.



Типы портов/технологий:

- 10/100Base-TX – стандарт Ethernet/Fast Ethernet на витой паре;
- 100/1000Base-T – стандарт Fast Ethernet/Gigabit Ethernet на витой паре;
- 1000Base-X SFP – стандарт Gigabit Ethernet на оптоволоконном кабеле;
- комбо-порт 100/1000Base-T/SFP – 2 разъема коммутатора, один для 100/1000Base-T другой для 1000Base-X/SFP; таким образом, можно выбрать режим работы данного порта (одновременно оба режима не поддерживаются).

Все порты поддерживают полнодуплексный режим работы.

Требуется:

1) рассчитать фабрику производительности коммутационной матрицы коммутатора.

Решение. Согласно характеристикам коммутатора, 16 портов 10/100Base-TX могут работать на максимальной скорости 100 Мбит/с каждый, 2 комбо-

порта 100/1000Base-T/SFP могут работать на максимальной скорости 1000 Мбит/с каждый. Скорость работы коммутационной матрицы должна быть не меньше суммарной производительности всех портов, следовательно, учитывая полнодуплексный режим работы портов (когда имеется 2 потока кадров через порт в противоположных направлениях), выполним расчет:

Скм=2×(16×100+ 2×1000)=7,2 Гбит/с

2) Вычислить максимальную скорость продвижения кадров для порта, указанного в варианте задания при условии использования кадров Ethernet с размером поля данных, указанном в варианте задания.

Решение.

Пусть исследуемый порт 10/100Base-TX, размер поля данных N=256 байтов.

Кадр Ethernet имеет следующую структуру (размеры полей даны в байтах):

8	6	6	2	N	4
Преамбула	Адрес приемника	Адрес источника	Длина	Данные	Контрольная сумма

Общая длина кадра *L = 8 + 6 + 6 + 2 + 256 + 4 = 282 байта=2256 бит*

Согласно стандартам, межкадровый интервал IPG равняется:

- на скорости 10 Мбит/с IPG=9,6 мкс;
- на скорости 100 Мбит/с IPG=0,96 мкс;
- на скорости 1000 Мбит/с IPG=0,096 мкс.

Один битовый интервал 1bt также зависит от скорости передачи и равняется:

- на скорости 10 Мбит/с 1bt=0,1 мкс;
- на скорости 100 Мбит/с 1bt=0,01 мкс;
- на скорости 1000 Мбит/с 1bt=0,001 мкс.

Рассчитаем время передачи кадра с учетом межкадрового интервала для скорости канала 100 Мбит/с (берем максимальную скорость исследуемого порта):

Tk = 2256 бит × 0,01 мкс + 0,96 мкс = 23,52 мкс.

Тогда максимальная скорость продвижения кадров для порта в одном направлении передачи:

Vk = 1 / 23,52 мкс = 42 517 кадров/с.

3) Рассчитать максимальную полезную производительность данного порта при условии его работы в полнодуплексном режиме.

Решение. *Рпол = 2 × 42 517 кадров/с × 256 байтов × 8 = 174,14 Мбит/с.*

Таким образом, исследуемый порт коммутатора может поддерживать передачу полнодуплексного пользовательского трафика размером в 174,14 Мбит/с.

8.2 Порядок проведения практической работы

8.2.1 Изучить основные теоретические сведения о функциональных характеристиках коммутаторов локальных компьютерных сетей, методику расчета базовых характеристик, разъяснения преподавателя.

8.2.2 Получив вариант заданий, изучить из источника характеристики коммутатора, указанного в варианте.

8.2.3 Рассчитать фабрику производительности коммутационной матрицы коммутатора.

8.2.4 Вычислить максимальную скорость продвижения кадров для порта, указанного в варианте задания при условии использования кадров Ethernet с размером поля данных, указанном в варианте задания.

8.2.5 Рассчитать максимальную полезную производительность данного порта при условии его работы в полнодуплексном режиме.

Примечание:

а) для расчетов принять 1 Гбайт = 1 000 Мбайт = 1 000 000 Кбайт = 1 000 000 000 байт;

б) в конфигурациях коммутаторов разных моделей применяются следующие обозначения портов и типов каналов: 10/100Base-TX – стандарт Ethernet/Fast Ethernet на витой паре; 10/100/1000Base-T – стандарт Ethernet/Fast Ethernet/Gigabit Ethernet на витой паре; 1000Base-T – стандарт Gigabit Ethernet на витой паре; 1000Base-X SFP – стандарт Gigabit Ethernet на оптоволоконном кабеле; 10GBase-X SFP – стандарт 10 Gigabit Ethernet на оптоволоконном кабеле; комбо-порт 100/1000Base-T/SFP – 2 разъема коммутатора, один для 100/1000Base-T другой для 1000Base-X/SFP (таким образом, можно выбрать режим работы данного порта, одновременно оба режима не поддерживаются).

Список использованных источников

1. С. А. Орлов. Организация ЭВМ и систем: Учебник для ВУЗов. Для бакалавров и магистров. 4-е издание. – СПб.: Питер, 2020. – 688 с.

2. Таненбаум Э. Архитектура компьютера / Э. Таненбаум, Т. Остин. – 6-е изд. – Санкт-Петербург: Питер, 2019. – 816 с.

3. Брайант Рэндал Э., О'Халларон Дэвид Р. Компьютерные системы: архитектура и программирование. 3-е изд. / пер. с англ. А. Н. Киселева. – М.: ДМК Пресс, 2022. – 994 с.

4. Основы теории вычислительных систем. Под ред. Майорова С.А. – М.: Высшая школа, 1978. – 408с.

5. Хамахер, К. Организация ЭВМ / К. Хамахер, З. Вранешич, С. Заки. – 5-е изд. – СПб. : Питер, 2003. – 848 с.

6. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для ВУЗов. Юбилейное издание. – СПб.: Питер, 2020. – 1008 с.

7. Урбанович, П. П. Компьютерные сети : учебное пособие / П. П. Урбанович, Д. М. Романенко. – Москва ; Вологда : Инфра-Инженерия, 2022. – 460 с.

2.6 УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО КУРСОВОМУ ПРОЕКТИРОВАНИЮ

ОГЛАВЛЕНИЕ

1 Общие положения

1.1 Принципы высокопроизводительных вычислений

Применение сложных, в том числе интеллектуальных, подходов к обработке данных, представленных мега-, гига- и терабайтами, требует особого внимания к обеспечению их высокопроизводительной обработки. В условиях стремительного наращивания объемов, данных многие стандартные вычислительные средства и традиционные алгоритмические подходы для такой сложной и затратной обработки в значительной степени непригодны. Сформулируем несколько основных принципов организации высокопроизводительных вычислений.

Первый принцип организации высокопроизводительных вычислений заключается в увеличении производительности обработки данных за счет совершенствования вычислительной эффективности алгоритмов, позволяя применять достаточно сложную обработку данных, но в некоторых случаях доступную даже для ПЭВМ со стандартными характеристиками.

Сегодня доступны различные вычислительные мощности. Вычислительные кластеры организуют как на базе стандартных недорогих ПЭВМ, объединенных в локальную вычислительную сеть, так и в специализированных центрах, оснащенных суперкомпьютерной многопроцессорной техникой. Поэтому второй принцип определяет целесообразность использования таких подходов к обработке данных, которые применимы как на дорогостоящей суперкомпьютерной технике, так и на недорогих ПЭВМ, объединенных в сети. При этом подразумевается, что высокопроизводительная обработка достигается двумя способами, которые могут быть использованы совместно.

Первый способ предполагает обработку исключительно за счет применения более высокопроизводительной вычислительной техники. Второй – адаптацию существующих подходов для возможности не только канонического параллельного, но и так называемого распределенно-параллельного исполнения. Обширный класс параллельных вычислений характеризуется возможностью одновременного решения одной вычислительной задачи путем ее декомпозиции. Очевидно, параллельные вычисления могут быть реализованы на одной ПЭВМ в многопроцессном режиме под управлением многозадачной операционной системы. Параллельные вычисления на нескольких вычислительных узлах (например, нескольких ПЭВМ, объединенных в локальную вычислительную сеть, или нескольких процессорах суперкомпьютера) терминологически относят к распределенным вычислениям. Именно поэтому здесь и далее, применяя термин распределенно-параллельные вычисления, будем иметь в виду вычислительный процесс, реализуемый не только как параллельный, но и как распределенный.

Практическая организация высокопроизводительных распределенно-параллельных вычислений с использованием указанных выше принципов требует ряда важных пояснений.

Организация распределенных вычислений возможна с использованием

множества различных подходов и архитектур. Однако при всем многообразии возможных вариантов принципиально отличают два различных класса построения вычислительных систем – системы с общей (разделяемой) памятью и системы с распределенной памятью.

К первому варианту построения вычислительных систем относят системы с симметричной архитектурой и симметричной организацией вычислительного процесса – SMP-системы (англ. Symmetric MultiProcessing). Поддержка SMP-обработки данных присутствует в большинстве современных ОС при организации вычислительных процессов на многопроцессных (многоядерных) ПЭВМ. Однако разделяемая память требует решения вопросов синхронизации и исключительного доступа к разделяемым данным, а построение высокопроизводительных систем в этой архитектуре затруднено технологически сложностями объединения большого числа процессоров с единой оперативной памятью.

Второй вариант предполагает объединение нескольких вычислительных узлов (ВУ) с собственной памятью в единой коммуникационной среде, взаимодействие между узлами в которой осуществляется путем пересылки сообщений. Причем такими ВУ могут быть как стандартные недорогие ПЭВМ, так и процессоры дорогостоящего суперкомпьютера. Такая архитектура построения вычислительной системы характеризуется бóльшими возможностями построения высокопроизводительных вычислительных систем и значительного масштабирования доступных вычислительных мощностей. Однако, в отличие от SMP-систем, здесь более высокая латентность (существенные накладные коммуникационные расходы), негативно влияющая на производительность системы и требующая более внимательной организации распределенно-параллельной обработки. Причем в случае построения вычислительного кластера на базе ПЭВМ такая латентность, как правило, существенно выше, чем при построении кластера на базе суперкомпьютера, за счет значительно более развитой коммуникационной среды.

В соответствии с классификацией архитектур вычислительных систем М. Флинна наибольшее распространение на практике получили две модели параллельных вычислений – Multiple Process / Program – Multiple Data (MPMD, множество процессов / программ, множество данных) и Single Process / Program – Multiple Data (SPMD, один процесс / программа, множество данных). Модель MPMD предполагает, что параллельно выполняющиеся процессы исполняют различные программы (процессы, потоки) на различных процессорах. Модель SPMD обусловливает работу параллельно выполняющихся программ (процессов, потоков), но исполняющих идентичный код при обработке отличных (в общем случае) массивов данных.

Очевидно, организацию распределенных вычислений целесообразно реализовать с использованием программ с параллельной обработкой данных на основе модели SPMD. Во-первых, это более практично из-за необходимости разрабатывать и отлаживать лишь одну программу, а во-вторых, такие программы, как правило, применимы и при традиционном последовательном исполнении, что расширяет области их практической применимости.

Кроме вышеизложенных особенностей организации высокопроизводительных вычислений в различных архитектурах и на основе различных моделей, при разработке алгоритмического обеспечения параллельной обработки необходимо: – определить фрагменты вычислений, которые могут быть исполнены параллельно;

– распределить данные по модулям локальной памяти отдельных ВУ;

– согласовать распределение данных с параллелизмом вычислений.

Важным является выполнение всех перечисленных условий, так как иначе значительные фрагменты вычислений не удастся представить, как параллельно исполняемые и реализация алгоритма в распределенно-параллельной архитектуре не позволит добиться роста производительности. Задачи распределения данных по модулям памяти и задача согласования распределения данных важны для обеспечения низкой латентности между ВУ и обеспечения возможностей масштабирования системы.

Помимо этого, при построении такого рода вычислительных системы важно иметь четко измеримые показатели их эффективности. Первым таким критерием можно назвать параллельное ускорение, которое показывает ускорение выполнения по сравнению с последовательным вариантом.

1.2 Две парадигмы программирования

Развитие фундаментальных и прикладных наук, технологий требует применения все более мощных и эффективных методов и средств обработки информации. В качестве примера можно привести разработку реалистических математических моделей, которые часто оказываются настолько сложными, что не допускают точного аналитического их исследования. Единственная возможность исследования таких моделей, их верификации (то есть подтверждения правильности) и использования для прогноза - компьютерное моделирование, применение методов численного анализа. Другая важная проблема - обработка больших объемов информации в режиме реального времени. Все эти проблемы могут быть решены лишь на достаточно мощной аппаратной базе, с применением эффективных методов программирования.

В настоящее время наблюдается стремительный рост развития вычислительной техники. Производительность современных компьютеров на много порядков превосходит производительность первых ЭВМ и продолжает возрастать заметными темпами. Увеличиваются и другие ресурсы, такие как объем и быстродействие оперативной и постоянной памяти, скорость передачи данных между компонентами компьютера и т.д. Совершенствуется архитектура ЭВМ.

Вместе с тем следует заметить, что уже сейчас прогресс в области микроэлектронных компонент сталкивается с ограничениями, связанными с фундаментальными законами природы. Вряд ли можно надеяться на то, что в ближайшее время основной прогресс в производительности электронно-вычислительных машин будет достигнут лишь за счет совершенствования их элементной базы. Переход на качественно новый уровень производительности

потребовал от разработчиков ЭВМ и новых архитектурных решений. 1.3 Две модели программирования: последовательная и параллельная

Традиционная архитектура ЭВМ была последовательной. Это означало, что в любой момент времени выполнялась только одна операция и только над одним операндом. Совокупность приемов программирования, структур данных, отвечающих последовательной архитектуре компьютера, называется моделью последовательного программирования. Ее основными чертами являются применение стандартных языков программирования (как правило, это ФОРТРАН-77, ФОРТРАН-90, C/C++), достаточно простая переносимость программ с одного компьютера на другой и невысокая производительность.

Появление в середине шестидесятых первого компьютера класса суперЭВМ, разработанного в фирме CDC Сеймуром Крэем, ознаменовало рождение новой - векторной архитектуры. Начиная с этого момента, суперкомпьютером принято называть высокопроизводительный векторный компьютер. Основная идея, положенная в основу новой архитектуры, заключалась в распараллеливании процесса обработки данных, когда одна и та же операция применяется одновременно к массиву (вектору) значений. В этом случае можно надеяться на определенный выигрыш в скорости вычислений Идея параллелизма оказалась плодотворной и нашла воплощение на разных уровнях функционирования компьютера.

Основными особенностями модели параллельного программирования являются высокая эффективность программ, применение специальных приемов программирования и, как следствие, более высокая трудоемкость программирования, проблемы с переносимостью программ.

1.3 Две парадигмы параллельного программирования

В настоящее время существуют два основных подхода к распараллеливаниювычислений. Это *параллелизм данных и параллелизм задач*. В англоязычной литературе соответствующие термины - *data parallel и message passing*. В основе обоих подходов лежит распределение вычислительной работы по доступнымпользователю процессорам параллельного компьютера. При этом приходится решать разнообразные проблемы. Прежде всего, это достаточно равномерная загрузка процессоров, так как если основная вычислительная работа будет ложиться на один из процессоров, мы приходим к случаю обычных последовательных вычислений, и в этом случае никакого выигрыша за счет распараллеливания задачи не будет. Сбалансированная работа процессоров - это первая проблема, которую следует решить при организации параллельных вычислений. Другая и не менее важная проблема - скорость обмена информацией между процессорами. Если вычисления выполняются на высокопроизводительных процессорах, загрузка которых достаточно равномерная, но скорость обмена данными низкая, основная часть времени будет тратиться впустую на ожидание информации, необходимой для дальнейшей работы данного процессора. Рассматриваемые парадигмы

программирования различаются методами решения этих двух основных проблем. Разберем более подробно параллелизм данных и параллелизм задач.

Параллелизм данных. Основная идея подхода, основанного на параллелизме данных, заключается в том, что одна операция выполняется сразу над всеми элементами массива данных. Различные фрагменты такого массива обрабатываются на векторном процессоре или на разных процессорах параллельной машины. Распределением данных между процессорами занимается программа. Векторизация или распараллеливание в этом случае чаще всего выполняется уже на этапе компиляции - перевода исходного текста программы в машинные команды. Роль программиста в этом случае обычно сводится к заданию опций векторной или параллельной оптимизации компилятору, директив параллельной компиляции, использованию специализированных языков для параллельных вычислений. Наиболее распространенными языками для параллельных вычислений являются Высокопроизводительный ФОРТРАН (High Performance FORTRAN) и параллельные версии языка С (это, например, С*). Более детальное описание рассматриваемого подхода к распараллеливанию содержит указание на следующие его основные особенности:

- обработкой данных управляет одна программа;
- пространство имен является глобальным, то есть для программиста существует одна единственная память, а детали структуры данных, доступа к памяти и межпроцессорного обмена данными от него скрыты;
- слабая синхронизация вычислений на параллельных процессорах, то есть выполнение команд на разных процессорах происходит, как правило, независимо и только лишь иногда производится согласование выполнения циклов или других программных конструкций - их синхронизация. Каждый процессор выполняет один и тот же фрагмент программы, но нет гарантии, что в заданный момент времени на всех процессорах выполняется одна и та же машинная команда;
- параллельные операции над элементами массива выполняются одновременно на всех доступных данной программе процессорах.

Видим, таким образом, что в рамках данного подхода от программиста не требуется больших усилий по векторизации или распараллеливанию вычислений. Даже при программировании сложных вычислительных алгоритмов можно использовать библиотеки подпрограмм, специально разработанных с учетом конкретной архитектуры компьютера и оптимизированных для этой архитектуры. Подход, основанный на параллелизме данных, базируется на использовании при разработке программ базового набора операций:

- *операции управления данными;*
- *операции над массивами в целом и их фрагментами;*
- *условные операции;*
- *операции приведения;*
- *операции сдвига;*

- *операции сканирования;*
- *операции, связанные с пересылкой данных.*

Рассмотрим эти базовые наборы операций.

Управление данными. В определенных ситуациях возникает необходимость в управлении распределением данных между процессорами. Это может потребоваться, например, для обеспечения равномерной загрузки процессоров. Чем более равномерно загружены работой процессоры, тем более эффективной будет работа компьютера.

Операции над массивами. Аргументами таких операций являются массивы в целом или их фрагменты (сечения), при этом данная операция применяется одновременно (параллельно) ко всем элементам массива. Примерами операций такого типа являются вычисление поэлементной суммы массивов, умножение элементов массива на скалярный или векторный множитель и т.д. Операции могут быть и более сложными - вычисление функций от массива, например. Условные операции

Эти операции могут выполняться лишь над теми элементами массива, которые удовлетворяют какому-то определенному условию. В сеточных методах это может быть четный или нечетный номер строки (столбца) сетки или неравенствонулю элементов матрицы.

Операции приведения. Операции приведения применяются ко всем элементам массива (или его сечения), а результатом является одно единственное значение, например, сумма элементов массива или максимальное значение его элементов.

Операции сдвига. Для эффективной реализации некоторых параллельных алгоритмов требуются операции сдвига массивов. Примерами служат алгоритмы обработки изображений, конечно-разностные алгоритмы и некоторые другие.

Операции сканирования. Операции сканирования еще называются префиксными/суффиксными операциями. Префиксная операция, например, суммирование выполняется следующим образом. Элементы массива суммируются последовательно, арезультат очередного суммирования заносится в очередную ячейку нового, результирующего массива, причем номер этой ячейки совпадает с числом просуммированных элементов исходного массива.

Операции пересылки данных. Это, например, операции пересылки данных между массивами разной формы (то есть имеющими разную размерность и разную протяженность по каждому измерению) и некоторые другие.

При программировании на основе параллелизма данных часто используются специализированные языки - CM FORTRAN, C*, FORTRAN+, MPP FORTRAN, Vienna FORTRAN, а также HIGH PERFORMANCE FORTRAN

(HPF).

Параллелизм задач. Стиль программирования, основанный на параллелизме задач подразумевает,что вычислительная задача разбивается на несколько относительно самостоятельных подзадач и каждый процессор загружается своей собственной подзадачей. Компьютер при этом представляет собой MIMD - машину. Как было отмечено ранее, аббревиатура MIMD обозначает в известной классификации архитектур ЭВМ компьютер, выполняющий одновременно множество различных операций над множеством, вообще говоря, различных и разнотипных данных. Для каждой подзадачи пишется своя собственная программа на обычном языке программирования, обычно это ФОРТРАН или С. Чем больше подзадач, тем большее число процессоров можно использовать, тем большей эффективности можно добиться. Важно то, что все эти программы должны обмениваться результатами своей работы, практически такой обмен осуществляется вызовом процедур специализированной библиотеки. Программист при этом может контролировать распределение данных между процессорами и подзадачами и обмен данными. Очевидно, что в этом случае требуется определенная работа для того, чтобы обеспечить эффективное совместное выполнение различных программ. По сравнению с подходом, основанным на параллелизме данных, данный подход более трудоемкий, с ним связаны следующие проблемы:

- повышенная трудоемкость разработки программы и ее отладки;
- на программиста ложится вся ответственность за равномерную загрузку процессоров параллельного компьютера;
- программисту приходится минимизировать обмен данными между задачами, так как пересылка данных - наиболее "времяёмкий" процесс;
- повышенная опасность возникновения тупиковых ситуаций, когда отправленная одной программой посылка с данными не приходит к месту назначения.

Привлекательными особенностями данного подхода являются большая гибкость и большая свобода, предоставляемая программисту в разработке программы, эффективно использующей ресурсы параллельного компьютера и, как следствие, возможность достижения максимального быстродействия.

Примерами специализированных библиотек для организации параллельной обработки, являются:

- OpenMP (Open Multi-Processing) — открытый стандарт для распараллеливания программ на языках Си, C++ и Фортран. Даёт описание совокупности директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью. Разработка спецификаций стандарта ведётся некоммерческой организацией OpenMP Architecture Review Board (ARB), в которую входят все основные производители процессоров, а также ряд суперкомпьютерных лабораторий и университетов. OpenMP реализует параллельные вычисления с помощью многопоточности, в которой ведущий (англ. master) поток создаёт набор ведомых потоков, и задача

распределяется между ними. Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами (количество процессоров/ядер не обязательно должно быть больше или равно количеству потоков);

- PVM (Parallel Virtual Machine, дословно виртуальная параллельная машина) — общедоступный программный пакет, позволяющий объединять разнородный набор компьютеров в общий вычислительный ресурс («виртуальную параллельную машину») и предоставляющий возможности управления процессами с помощью механизма передачи сообщений. PVM — разработка Окриджской Национальной Лаборатории, университетов штата Теннеси и Эмори (Атланта). Существуют реализации PVM для самых различных платформ.

- MPI (Message Passing Interface, интерфейс передачи сообщений) — программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу. Библиотека MPI разработана в Аргоннской Национальной Лаборатории (США). MPI является наиболее распространённым стандартом интерфейса обмена данными в параллельном программировании, существуют его реализации для большого числа компьютерных платформ. Используется при разработке программ для кластеров и суперкомпьютеров. Основным средством коммуникации между процессами в MPI является передача сообщений друг другу.

2 MPI – Message Passing Interface

MPI – это большая библиотека функций для передачи сообщений, которая позволяет пользователю переделать свою последовательную программу в программу, которая будет полностью использовать параллельную архитектуру используемой вычислительной системы. MPI предоставляет программисту единый механизм взаимодействия ветвей внутри параллельного приложения независимо от машинной архитектуры (однопроцессорные/многопроцессорные с общей памятью / раздельной памятью), взаимного расположения ветвей (на одном процессоре / на разных процессорах).

Программа, использующая MPI, легче отлаживается и быстрее переносится на другие платформы. Минимально в состав MPI входят библиотека программирования (заголовочные и библиотечные файлы для языков C, C++ и Фортран) и загрузчик приложений.

Для MPI необходимо создавать приложения, содержащие код всех вервей сразу. MPI-загрузчиком запускается указываемое количество экземпляров программы. Каждый экземпляр определяет свой порядковый номер в запущенном коллективе, и в зависимости от этого номера и размера коллектива выполняет ту или иную ветвь алгоритма. Такая модель параллелизма называется Single Program/Multiple Data (SPMD) и является частным случаем модели Multiple Instruction/Multiple Data (MIMD). Каждая ветвь имеет пространство данных, полностью изолированное от других ветвей. Обмениваются данными

ветви только в виде сообщений MPI. Все ветви запускаются загрузчиком одновременно. Количество ветвей фиксировано – это означает, что в ходе работы порождение новых вервей невозможно.

Разные MPI-процессы могут выполняться как на разных процессорах, таки на одном и том же – для MPI-программы это роли не играет, поскольку в обоих случаях механизм обмена данными одинаков. Процессы обмениваются друг с другом данными в виде сообщений. Сообщения проходят под идентификаторами, которые позволяют программе и библиотеке связи отличать их друг от друга. Для совместного проведения тех или иных расчетов процессы внутри приложения объединяются в группы. Каждый процесс может узнать у библиотеки связи свой номер внутри группы, и, в зависимости от номера, приступает к выполнению соответствующей части расчетов. MPI-ветвь запускается и работает как обычный процесс, связанный через MPI с остальными процессами, входящими в приложение. В остальном процессы следует считать изолированными друг от друга – у них разные области кода, стека и данных. Особенностью MPI является введение понятия *области связи (communication domains).* При запуске приложения все процессы помещаются в создаваемую для приложения общую область связи. При необходимости они могут создавать новые области связи на базе существующих. Все области связи имеют независимую друг от друга нумерацию процессов. Программе пользователя в распоряжение предоставляется *коммуникатор* – описатель области связи. Многие функции MPI имеют среди входных аргументов коммуникатор, который ограничивает сферу их действия той областью связи, к которой он прикреплен. Для одной области связи может существовать несколько коммуникаторов таким образом, что приложение будет работать с ней как с несколькими разными областями. Так, в исходных текстах примеров для MPI часто используется идентификатор **MPI_COMM_WORLD**. Это название коммуникатора, создаваемого библиотекой автоматически. Он описывает стартовую область связи, объединяющую все процессы приложения. Далее в материале рассмотрены наиболее употребительные функции MPI.

Наиболее общие функции MPI и правила их использования

Существует несколько категорий функций:

- блокирующие;
- локальные;
- коллективные.

Блокирующие – останавливают (блокируют) выполнение процесса до тех пор, пока производимая ими операция не будет выполнена. *Неблокирующие* функции возвращают управление немедленно, а выполнение операции продолжается в фоновом режиме; за завершением операции надо проследить особо. Неблокирующие функции возвращают *квитанции (requests),* которые погашаются при завершении. До погашения квитанции с переменными и массивами, которые были аргументами не блокирующей функции, ничего делать нельзя. *Локальные* функции не инициируют пересылок данных между ветвями. Большинство информационных функций является локальными, т.к. копии

системных данных уже хранятся в каждой ветви. Например, рассматриваемые ниже функция передачи **MPI_Send** и функция синхронизации **MPI_Barrier** не являются локальными, поскольку производят пересылку. Следует заметить, что, к примеру, функция приема **MPI_Recv** (парная для **MPI_Send**) является локальной: она всего лишь пассивно ждет поступления данных, ничего не пытаясь сообщить другим ветвям. *Коллективные* – должны быть вызваны всеми ветвями–абонентами того коммуникатора, который передается им в качестве аргумента. Несоблюдение для них этого правила приводит к ошибкам на стадии выполнения программы (как правило, к повисанию) [5].

Правила использования идентификаторов MPI. Все идентификаторы начинаются с префикса "MPI_". Это правило без исключений. Не рекомендуется заводить пользовательские идентификаторы, начинающиеся с этой приставки, а также с приставок "MPID_", "MPIR_" и "PMPI_", которые используются в служебных целях. Если идентификатор сконструирован из нескольких слов, слова в нем разделяются подчерками: MPI_Get_count, MPI_Comm_rank. Иногда, однако, разделитель не используется: MPI_Sendrecv, MPI_Alltoall. Порядок слов в составном идентификаторе выбирается по принципу "от общего к частному": сначала префикс "MPI_", потом название категории (Type, Comm, Group, Attr, Errhandler и т.д.), потом название операции (MPI_Errhandler_create, MPI_Errhandler_set,...). Наиболее часто употребляемые функции выпадают из этой схемы: они имеют "анти–методические", но короткие и стереотипные названия, например MPI_Barrier, или MPI_Unpack. Имена констант (и неизменяемых пользователем переменных) записываются полностью заглавными буквами: MPI_COMM_WORLD, MPI_FLOAT. В именах функций первая за префиксом буква – заглавная, остальные маленькие: MPI_Send, MPI_Comm_size.

Существует набор функций, которые используются в любом, даже самом коротком приложении MPI. Занимаются они не столько собственно передачей данных, сколько ее обеспечением. Ниже рассматривается ряд наиболее употребимых функций этого класса.

Подпрограммы управления средой MPI. Первое, что параллельная программа должна делать в течение времени выполнения, состоит в том, чтобы создать параллельную среду. Это означает, что она резервирует и устанавливает N узлов, на которых программа должна выполниться, и инициализирует MPI среду. Количество узлов (процессов) N входит в программу как системная переменная. Ниже в подразделе приведены наиболее важные из них, используемые практически в любом приложении MPI.

```
MPI_Init(&argc, &argv)
```

Данная функция обеспечивает инициализацию библиотеки. Является одной из первых инструкций в функции main (главной функции приложения). Она

получает адреса аргументов, стандартно получаемых самой main от операционной системы и хранящих параметры командной строки.

MPI_Finalize()

Нормальное закрытие библиотеки. Никакая другая MPI функция не может быть вызвана после этого. Настоятельно рекомендуется не забывать вписывать эту инструкцию перед возвращением из программы, т.е.:

- перед вызовом стандартной функции языка C – exit ;
- перед каждым после MPI_Init оператором return в функции main;
- если функции main назначен тип void, и она не заканчивается оператором return, то MPI_Finalize() следует поставить в конец main.

Пример 1 демонстрирует правила применения описанных функций при организации MPI-приложения.

Пример 1

```
#include "mpi.h"
#include <stdio.h>
/* mpi.h обеспечивает основные определения и типы MPI */
int main(argc, argv)
int argc; char **argv;
{
MPI_Init(&argc, &argv); printf("Hello world\n");MPI_Finalize();
return 0;
}
```

MPI_Abort (comm, error)

Вызов MPI_Abort из любой задачи принудительно завершает работу ВСЕХ задач, подсоединенных к заданной области связи. Если указан описатель области связи comm в виде MPI_COMM_WORLD, будет завершено все приложение (все его задачи) целиком, что, по-видимому, и является наиболее правильным

решением. Если неизвестно, как охарактеризовать ошибку error в классификации MPI, рекомендуется использовать код ошибки MPI_ERR_OTHER.

Следующие две информационных функции сообщают размер группы (т.е. общее количество задач, подсоединенных к ее области связи) и порядковый номер вызывающей задачи.

MPI_Comm_size (comm, *size)

Число процессов возвращает функция MPI_Comm_size. size – это число всех процессов в пределах некоторой группы, названной comm. Обычно comm

является предопределенной коммуникатором MPI_COMM_WORLD, который включает все процессы в прикладную программу MPI.

MPI_Comm_rank (comm, *rank)

Каждый процесс характеризуется собственным целочисленным идентификатором называемым рангом (rank). Ранги непрерывны и начинаются с нуля и заканчиваются size–1. Процесс с нулевым рангом обычно называется ГЛАВНЫМ или MASTER процессом. Функция MPI_Comm_rank возвращает ранг вызываемого процесса. Пример 2 демонстрирует использование MPI_Comm_size и MPI_Comm_rank.

Пример 2

```
#include "mpi.h"
#include <stdio.h>

int main(argc, argv)
int argc;
char **argv;
{
int rank, size; MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
printf("Hello world! I'm %d of %d\n", rank, size);
MPI_Finalize();
return 0;
}
```

MPI_Wtime()

Функция возвращает затраченное время (в секундах с двойной точностью) на вызов процесса. *Эта функция полезна, чтобы проверить время выполнения программы.* Пример 3 показывает, как использовать приведенные функции.

Пример 3

```
#include "mpi.h"
#include   <stdio.h>

void main(int argc, char *argv [])
{
int numtasks, rank, rc;
rc = MPI_Init(&argc, &argv);
if (rc!= 0) {
```

```
printf (" Ошибка при запуске MPI программы ");
printf(".Завершение\n ");
MPI_Finalize();
}
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
printf(" Число задач =%d Мой ранг =%d\n", numtasks, rank);
printf(" Время начала: % lf \n ", MPI_Wtime());

/******* делает некоторую работу *******/

printf(" Время конца: %lf \n ", MPI_Wtime());
MPI_Finalize();
}
```

В языке программирования C все MPI-подпрограммы после завершения работы возвращают целое число. Если возвращаемое значение ноль – успешное завершение работы, в противном случае – ненулевое значение. Эта информация может использоваться для обработки ошибок как в примере 3, рассмотренном выше.

Процедуры MPI передачи сообщений. Связь между задачами возможна через многие MPI подпрограммы. Здесь внимание сосредоточено на трех подпрограммах, которые являются наиболее полезными для разработки параллельных приложений.

Функция MPI_Send рассылает данные из буфера buf, количество данных count, тип – datatype. Ниже приведен синтаксис этой функции.

MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Параметры
buf – адрес буфера с информацией;
count – количество элементов в буфере;
datatype – тип элемента в буфере;
dest – ранг процесса, которому посылаются данные;
tag – таг сообщения;
comm – дескриптор.

MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

Функция MPI_Recv получает данные.

Параметры

buf – адрес буфера с информацией;

count – максимальное количество элементов в буфере;

datatype – тип элемента в буфере;

tag – таг сообщения;

comm – дескриптор;

status – объект статуса;

source – ранг процесса, который посылает сообщение.

Параметр dest (source) определяет процесс, которому должно быть доставлено (из которого отправлено) сообщение и определяется как ранг процесса, который получает (отправляет) сообщение. Каждое сообщение помечается своей уникальной целочисленной отметкой (tag). Параметр tag опознает сообщение и должен соответствовать одной из получаемых задач. Для получателя символ MPI_ANY_TAG может использоваться, чтобы получить любое сообщение независимо от его отметки (tag). Наконец, операция указывает на источник и отметку полученного сообщения. В Си этот параметр обозначает указатель на структуру MPI_STATUS (stat. MPI_SOURCE, stat. MPI_TAG).

Следует заметить, что параметр count содержит максимальное количество элементов в буфере. Его значение можно определить с помощью MPI_Get_count.

Все типы в MPI предопределены. Наиболее употребимые из них приведены в таблицу 1.

Таблица 1 – типы данных MPI

MPI datatype	C datatype
MPI CHAR	signed char
MPI SHORT	signed short int
MPI INT	signed int
MPI LONG	signed long int
MPI UNSIGNED CHAR	unsigned char
MPI UNSIGNED SHORT	unsigned short int
MPI UNSIGNED	unsigned int
MPI UNSIGNED LONG	unsigned long int
MPI FLOAT	float
MPI DOUBLE	double
MPI LONG DOUBLE	long double
MPI PACKED	

Замечание. Все MPI функции (за исключением MPI_Wtime и MPI_Wtick) возвращают информацию об ошибках. В функциях системы программирования С – это возвращаемое значение функции. Перед тем, как функция завершит свою работу, вызывается текущий обработчик ошибок. По умолчанию он прекращает работу этой функции. Значение обработчика ошибок может быть изменено в

MPI_Errhandler_set. Значения, которые может приобретатьобработчик ошибок, сведены в таблицу 2.

Таблица 2 – Значения обработчика ошибок

Возвращаемое значение функции	Значения обработчика ошибок
MPI_SUCCESS	Нет ошибок. MPI функция завершилась успешно
MPI_ERR_COMM	Недопустимый коммуникатор. В вызове необходимо использовать нулевой коммуникатор.
MPI_ERR_COUNT	Недопустимый параметр count. Параметр count должен быть положительным или нулевым.
MPI_ERR_TYPE	Неправильный тип параметра.
MPI_ERR_TAG	Неправильный параметр тег. Теги должны бать положительными. В функциях MPI_Recv, MPI_Irecv, MPI_Sendrecv, и т.п. они могут принимать значение MPI_ANY_TAG. Максимальное значение тегаможно получить через MPI_TAG_UB.
MPI_ERR_RANK	Недопустимое значение ранга. Ранг должен находиться в границах от 0до size–1 (size – размер коммуникатора). В функциях MPI_Recv, MPI_Irecv, MPI_Sendrecv, и т.п. он может принимать значение MPI_ANY_SOURCE.

Последний тип MPI_PACKED не соответствуют стандартному типу языка С. Правила его использования описаны далее.

Пример 4 показывает использование MPI_Send и MPI_Recv. Приведенная программа берет данные нулевого процесса и рассылает их всем остальным процессам по кругу. Это означает, что процесс *i* должен получить данные от процесса *i–1* и переслать их процессу *i+1*.

Пример 4

```
#include <stdio.h>
#include "mpi.h"

int main(argc, argv)
int argc;
char **argv;
{
int rank, value, size;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
do {
if (rank == 0) {
scanf("%d", &value);
MPI_Send(&value, 1, MPI_INT, rank + 1, 0,MPI_COMM_WORLD);
}
else {
MPI_Recv(&value, 1, MPI_INT, rank - 1, 0,MPI_COMM_WORLD, &status);
if (rank < size - 1)
  MPI_Send(&value, 1, MPI_INT, rank + 1, 0,MPI_COMM_WORLD);
}
printf("Process %d got %d\n", rank, value);
}
while (value >= 0);
MPI_Finalize();
return 0;
}
```

Рассмотрим еще одну программу с использованием этих функций (пример 5). Здесь проверяется, правильно ли рассылаются и получаются сообщения. В этой программе все процессы (кроме нулевого) посылают 3 сообщения нулевому процессу. Нулевой процесс печатает это сообщение, как только он его получает (используем MPI_ANY_SOURCE и MPI_ANY_TAG в MPI_Recv).

Пример 5

```
#include "mpi.h"
#include <stdio.h>>

int main(argc, argv)
int argc;
char **argv;
{
int rank, size, i, buf[1];
MPI_Status status;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
if (rank == 0) {
for (i=0; i<100*(size–1); i++) {
MPI_Recv( buf, 1, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &status );
printf( "Msg from %d with tag %d\n",
status.MPI_SOURCE, status.MPI_TAG );
```

```
}
}
else {
for (i=0; i<100; i++)
MPI_Send( buf, 1, MPI_INT, 0, i, MPI_COMM_WORLD );
}
MPI_Finalize();
return 0;
}
```

MPI_Bcast (void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Функция MPI_Bcast пересылает информацию с процесса root ко всем остальным.

Параметры
buf – адрес буфера с информацией;
count – количество элементов в буфере;
datatype – тип элемента в буфере;
root – ранг root процесса;
comm – дескриптор.

Здесь root – это ранг процесса, который рассылает данные. Часто требуется, чтобы один из процессов читал данные с клавиатуры или командной строки, а потом рассылал эту информацию всем остальным процессам.

Рассмотрим программу, которая читает целое число, вводимое пользователем, и рассылает это его остальным процессам (пример 6). Каждый процесс должен напечатать свой ранг и значение, которое он получил. Значения вводятся, пока не будет введено отрицательное число.

Пример 6

```
#include <stdio.h>
#include "mpi.h"

int main( argc, argv )
int argc;
char **argv;
{
int rank, value;
MPI_Init( &argc, &argv );
```

```
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
do {
if (rank == 0)
scanf( "%d", &value );
MPI_Bcast( &value, 1, MPI_INT, 0, MPI_COMM_WORLD );
printf( "Process %d got %d\n", rank, value );
}
while (value >= 0);
MPI_Finalize( );return 0;
}
```

MPI_Reduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

С командой MPI_Reduce узел root собирает данные от всех узлов, включая себяи применяет к ним операцию op.

Параметры
sendbuf – адрес буфера с информацией;
count – количество элементов в буфере;
datatype – тип элемента в буфере;
op – операция;
root – ранг главного процесса;
recvbuf – адрес буфера, который будет принимать данные;
comm – дескриптор.

sendbuf содержит данные, которые должны быть посланы, а recvbuf получает их. Операцией op (operation) выполняется заданная операция над данными. Наиболее общие операции предопределены как:
MPI_MAX – максимум;
MPI_MIN – минимум;
MPI_SUM – суммирование;
MPI_PROD – программа.

Обычно процесс сбора – MASTER процесс, т.е процесс с нулевым рангом. Рассмотрим пример 7, в котором вычисляется значение □. Подсчитывается интеграл *4/(1+x*x)* в промежутке от *–1/2* и *1/2*. Алгоритм простой: интеграл аппроксимируется суммой *n* интервалов. Аппроксимация интеграла на каждом интервале – *(1/n)*4/(1+x*x)*. Главный процесс (ранг 0) делает запрос на введения количества интервалов. Далее он рассылает это число всем процессам. Каждый процесс складывает *n* интервалов *(x=–1/2+rank/n, –1/2+rank/n+size/n,...).* В конце, суммы, вычисляемые каждым процессом, складываются вместе.

Пример 7

```
#include "mpi.h"
 #include <stdio.h>
int main(argc,argv)
int argc;
char *argv[];
{
int done = 0, n, myid, numprocs, i;
double PI25DT = 3.14159265358979323846264 3;
double mypi, pi, h, sum, x;

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
while (!done)
{
if (myid == 0) {
printf("Enter the number of intervals:(0 quits) ");
scanf("%d",&n);
}
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
if (n == 0) break;
h= 1.0 / (double) n;
sum = 0.0;
for (i = myid + 1; i <= n; i += numprocs)
{
x = h * ((double)i – 0.5);
sum += 4.0 / (1.0 + x*x);
}
mypi = h * sum;
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,MPI_COMM_WORLD);
if (myid == 0)
printf("pi is approximately %.16f, Error is%.16f\n",
}
MPI_Finalize();
return 0;
}
```

MPI_Gather(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int dest, MPI_Comm comm)

Функция *MPI_Gather* собирает в приемный буфер задачи dest

передающиебуфера остальных задач

sbuf – адрес начала буфера посылки;
scount – число элементов в посылаемом сообщении;
stype – тип элементов отсылаемого сообщения;
OUT rbuf – адрес начала буфера сборки данных;
rcount – число элементов в принимаемом сообщении;
rtype – тип элементов принимаемого сообщения;
dest – номер процесса, на котором происходит сборка данных;
comm – идентификатор группы.

MPI_Barrier(MPI_Comm comm)

Является функцией синхронизации – останавливает выполнение вызвавшей ее задачи до тех пор, пока не будет вызвана изо всех остальных задач, подсоединенных к указываемому коммуникатору. Гарантирует, что к выполнению следующей за MPI_Barrier инструкции каждая задача приступит одновременно с остальными. Данная функция является единственной, вызовами которой гарантированно синхронизируется во времени выполнение различных ветвей. Некоторые другие коллективные функции в зависимости от реализации могут обладать, а могут и не обладать свойством одновременно возвращать управление всем ветвям. Однако для них это свойство является побочным и необязательным. Поэтому, если в программе нужна синхронность, необходимо использовать только MPI_Barrier.

Приведенные ниже 4 функции рассмотрены более детально в примере 8.

MPI_Address(void *location, MPI_Aint *address)

Получает адрес параметра в памяти (address). Во многих системах адрес, возвращаемый этой функцией аналогичен оператору & в Си.

MPI_Type_Struct

Создает структурный тип данных

MPI_Type_Commit (MPI_datatype *datatype)

Передает тип данных.

MPI_Datatype_free (MPI_Datatype *datatype)

Освобождает тип данных.

Рассмотрим в примере 8 использование 4 последних функций. В программе

примера процессу 0 передаются целое и действительное числа. Из них создается структура данных, которая функцией MPI_Bcast рассылается всем остальным процессам. Ввод завершается при введении отрицательного целого.

Пример 8

```
#include <stdio.h>
#include "mpi.h"
int main(argc, argv)
int argc;
char **argv;
{
int rank;
struct { int a; double b } value;
MPI_Datatype mystruct;
Int blocklens[2];
MPI_Aintindices[2];
MPI_Datatype old_types[2];

MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
/* По одному значению каждого типа */blocklens[0] = 1;
blocklens[1] = 1;
/* Типы параметров */
old_types[0] = MPI_INT;
old_types[1] = MPI_DOUBLE;
/* Расположение каждого элемента */
MPI_Address( &value.a, &indices[0] );
MPI_Address( &value.b, &indices[1] );
/* Делает зависимыми */
indices[1] = indices[1] - indices[0];indices[0] = 0;
MPI_Type_struct( 2, blocklens, indices, old_types,&mystruct );
MPI_Type_commit( &mystruct );

do {
if (rank == 0)
scanf( "%d %lf", &value.a, &value.b );
MPI_Bcast( &value, 1, mystruct, 0, MPI_COMM_WORLD );
printf( "Process %d got %d and %lf\n", rank, value.a,value.b );
} while (value.a >= 0);

/* Удаление типа*/
MPI_Type_free( &mystruct );
```

```
MPI_Finalize( );
return 0;
}
```

MPI_Packint MPI_Pack (void *inbuf, int incount, MPI_Datatype datatype, void *outbuf, int outcount, int *position, MPI_Comm comm)

Упаковывает тип данных в непрерывную область памяти.

Параметры
входные:
inbuf – начало буфера;
incount – количество вводимых элементов;
datatype – тип вводимых элементов;
outcount – выходной размер буфера (в байтах);
position – текущая позиция в буфере (в байтах);
comm – коммуникатор для упакованных сообщений; выходные:
outbuf – конец буфера.

MPI_Unpackint MPI_Unpack (void *inbuf, int insize, int *position, void *outbuf, int outcount, MPI_Datatype datatype, MPI_Comm comm)

Распаковывает тип данных.

Параметры
входные:
inbuf – начало буфера;
insize – размер буфера (в байтах);
position – текущая позиция в буфере (в байтах);
outcount – количество элементов, что должны быть распакованы;
datatype – тип выходных элементов;
comm – коммуникатор для упакованных данных; выходные:
outbuf – конец буфера.

Рассмотрим предыдущую программу с использованием двух последних функцийв примере 9. Используем MPI_Pack для упаковки данных в буфер (для простотыиспользуем packbuf[100]). Следует заметить, что MPI_Bcast, в отличие от MPI_Send/MPI_Recv, требует, чтобы одинаковое количество данных было послано и получено. Таким образом, необходимо удостовериться, что все процессы имеют одно и тоже значение параметра count в MPI_Bcast.

Пример 9.

```
#include <stdio.h>
#include "mpi.h"
int main( argc, argv )
int argc;
char **argv;
{
Int rank;
Int packsize, position;
int a;
double b;
char packbuf[100];
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
do {
if (rank == 0) {
scanf( "%d %lf", &a, &b );packsize = 0;
MPI_Pack( &a, 1, MPI_INT, packbuf, 100, &packsize,MPI_COMM_WORLD );
MPI_Pack( &b, 1, MPI_DOUBLE, packbuf, 100, &packsize,
MPI_COMM_WORLD);

}
MPI_Bcast( &packsize, 1, MPI_INT, 0, MPI_COMM_WORLD );
MPI_Bcast( packbuf, packsize, MPI_PACKED, 0,
MPI_COMM_WORLD );
if (rank != 0) {
position = 0;
MPI_Unpack( packbuf, packsize, &position, &a, 1,  MPI_INT,
MPI_DOUBLE, MPI_COMM_WORLD );
}
MPI_Unpack( packbuf, packsize, &position, &b, 1, MPI_COMM_WORLD);
printf( "Process %d got %d and %lf\n", rank, a, b );
} while (a >= 0);
MPI_Finalize( );
return 0;
}
```

3 Основные этапы курсового проектирования

Название курсового проекта

Реализация параллельной обработки для кластерных / мультипроцессорных систем на базе протокола MPI.

Цель курсового проектирования

Приобретение навыков по разработке и реализации параллельных алгоритмов для многокомпонентных систем обработки данных.

Разработка параллельных приложений использованием интерфейса MPI.

Сравнительный анализ временных характеристик выполнения обработки данных.

Варианты задач разработки

Обработка графических изображений, задачи комбинаторной оптимизации, машинное обучение, обработка больших данных (конкретные варианты заданий формируются руководителем курсового проекта).

Конфигурирование MPI-окружения и запуск приложения (на примере библиотеки WMPI for WINDOWS)

Для организации работы параллельного MPI-приложения необходимо сконфигурировать файл MPI-окружения <имя_файла_приложения>.pg и разместить его в каталоге приложения. При запуске программы происходит обращение к файлу с расширением pg, который должен иметь такое же имя, как и имя файла, содержащего код программы. В нем должно быть указано количество процессов, которые необходимо создать и адреса рабочих станций, на которых процессы должны быть созданы. Например, файл test.pg содержит следующую информацию:

```
local 3
c310_1 2 g:\kss\test.exe
```

Слово local означает, что указанные после него процессы будут созданы на той рабочей станции, на которой была запущена программа. При создании процессов на других рабочих станциях указывается адрес рабочей станции, количество создаваемых процессов, путь к каталогу, в котором располагается файл с кодом программы (вторая строка, приведенного в примере файла);

Исходные данные

Имеется один из вариантов компьютерных систем, аппаратная организация которых допускает параллельную обработку:

1) многокомпонентная вычислительная система, состоящая из вычислительных модулей (ВМ), каждый из которых имеет память для хранения обрабатываемых данных и результатов (например, кластер рабочих станций в составе локальной вычислительной сети) – как вариант распределённой архитектуры;

2) мультипроцессорная (мульти-ядерная) компьютерная система – как вариант сосредоточенной архитектуры.

Известно, *что время выполнения одной арифметической операции намного меньше, чем время передачи одного числа от ВМ к ВМ*. Известно также, что

выгоднее по времени передавать большие блоки данных, причем, чем больше пакет, тем лучше.

Дополнительно задаются следующие исходные данные:

- описание задачи обработки данных (согласно варианта задания);
- описание размерности решаемой задачи (согласно варианта задания);
- количество процессов, необходимых для реализации параллельного алгоритма (согласно варианта задания);
- язык программирования – С ++;
- MPI-библиотека.

Замечание**:** для выполнения раздела 5 курсового проекта необходимо реализовать в программах средства для проведения исследований по оцениванию времени выполнения алгоритма с учетом и без учета межпроцессных пересылок MPI.

Порядок выполнения курсового проекта

- на основе анализа варианта задания разработать последовательный алгоритм обработки;
- разработать программную систему, реализующую последовательный алгоритм, используя язык программирования С ++;
- изучить основы программирования задач с использованием интерфейса MPI;
- выполнить анализ последовательного алгоритма с целью выделения независимых по данным и управлению фрагментов, обработку которых возможно организовывать в параллельном режиме. Составить варианты схем параллелизации, обосновать выбор одного из них для последующей реализации на основе MPI. *При выборе варианта крайне необходимо учитывать тот факт, что каждая межмодульная пересылка сопровождается значительными временными издержками, связанными с организацией канала пересылки. Следовательно, целесообразно минимизировать количество межмодульных пересылок и увеличивать объем пересылаемых данных для каждой пересылки.* Для выбранноговарианта составить временную диаграмму этапов параллельной обработки и этапов межмодульного обмена;
- разработать алгоритм, реализующий выбранную схему параллелизации; на основе последовательной программы разработать программную систему, реализующую параллельный алгоритм, используя язык программированияС++ и MPI. При этом, с целью анализа производительности программы, обеспечить систему средствами, позволяющими оценить реальное время обработки;
- выполнить тестирование и рассчитать среднюю производительность разработанных программ с последовательной ($\lambda_1=1/T_1$) и параллельной ($\lambda_p=1/T_p$) обработкой на исходных данных заданной размерности (согласно варианту задания), T_1, T_p - фактическое среднее время последовательной и параллельной программы. Для этого рассчитать средние значения времен выполнения процессов по результатам нескольких (не менее 10) экспериментов; при этом зафиксировать как наблюдаемые, так и средние значения в сводной таблице;
- на основе результатов предыдущего этапа рассчитать коэффициент увеличения производительности (ускорения) за счет параллелизации:

$$m=\lambda_p/\lambda_1= E_1/T_p$$

Сделать выводы относительно эффективности разработанной схемы параллелизации.

Требования к содержанию разделов пояснительной записки

Во введении рассматриваются и анализируются (в краткой форме) цель и задачи курсового проекта, особенности прикладной задачи, требующей решения в процессе выполнения курсового проекта.

В первом разделе выполняется детальный анализ алгоритмов решения задачи, приводятся форматы исходных данных; анализируются особенности реализации алгоритмов обработки на языке программирования с учетом размерности исходных данных, размеров выделяемых буферов и других факторов, обуславливающих особенности реализуемых алгоритмов. Приводится схема алгоритма и ее детальное описание.

Во втором разделе приводятся особенности реализации алгоритма на языке C++, описание ключевых процедур решения задачи.

Во третьем разделе на основе анализа последовательного алгоритма должны быть предложены и детально описаны варианты схем параллелизации алгоритма с учетом имеющегося ассортимента функций MPI; выполнен анализ вариантов с точки зрения заданного количества процессов, объема передаваемых данных между процессами, трудоемкостью параллелизуемых ветвей; обоснованно выбран в качестве базового вариант схемы. Приводится выбранная схема алгоритма параллельной обработки.

В четвертом разделе приводится детальное описание структуры системы, программных модулей системы параллельной обработки с использованием MPI, описание и особенности использования конкретных функций MPI с приведением точного размера и типа передаваемых данных между процессами; допускается вставлять в текст раздела фрагменты программы с комментариями. Разрабатывается и приводится схема ресурсов системы параллельной обработки.

В пятом разделе детально описываются численные эксперименты по анализу временных характеристик выполнения приложения; должны быть приведены условия проведения исследований, формулы и результаты расчетов с комментариями. Численные данные свести в таблицу. Приводятся скриншоты результатов функционирования программ, в том числе численных.

В заключении кратко подводится итог результатам выполнения заданий курсового проекта.

Приложение должно содержать распечатку текстов программ последовательной и параллельной обработки; в приложения могут быть включены иные материалы по желанию разработчика.

Список использованных источников

1. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. -М.: Изд-во МГУ, 2004.-71с.

2. С. А. Орлов. Организация ЭВМ и систем: Учебник для ВУЗов. Для бакалавров и магистров. 4-е издание. – СПб.: Питер, 2020. – 688 с.

3. Таненбаум Э. Архитектура компьютера / Э. Таненбаум, Т. Остин. – 6-е изд. – Санкт-Петербург: Питер, 2019. – 816 с.

4. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для ВУЗов. Юбилейное издание. – СПб.: Питер, 2020. – 1008 с.

3 РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

3.1 Экзаменационный материал

Курсовое проектирование

Защита курсового проекта состоит из двух этапов:

- проверка пояснительной записки и разработанного программного обеспечения;
- проверка теоретических знаний по тематике курсового проекта методом тестирования.

Список тестовых вопросов и вариантов ответов с целью подготовки приведен ниже. В каждом тестовом вопросе правильными могут являться один, два или три варианта ответов; таким образом, положительным ответом на тестовый вопрос является указание студентом соответствующей комбинации правильных ответов.

Тестовые вопросы для защиты курсового проекта по дисциплине «Компьютерные системы и сети» по теме «Реализация параллельной обработки для кластерных / мультипроцессорных систем на базе протокола MPI»

Вопрос № 1-40010117

Последовательная модель программирования это:

Ответ:

(1) подход к разработке программного обеспечения, ориентированный на традиционную фон-неймановскую архитектуру вычислительных систем

(2) подход к разработке программного обеспечения, основанный на систематическом применении одного и того же языка программирования

(3) модель программирования, ориентированная на SISD-компьютеры по классификации Флинна

Вопрос № 2-40010118

Параллельная модель программирования это

Ответ:

(1) подход к разработке программного обеспечения, основанный на декомпозиции задачи и распределении вычислительной работы между разными вычислительными узлами и/или ядрами процессора

(2) подход к разработке программного обеспечения, состоящий в том, что одновременно разрабатываются несколько различных версий одной программы

(3) модель программирования, ориентированная, в том числе, и на MIMD-компьютеры по классификации Флинна

Вопрос № 3- 40010119

Параллельное программирование с использованием MPI имеет дело с параллелизмом на уровне:

Ответ:

(1) задач

(2) заданий

(3) машинных команд

Вопрос № 4-40010120

Особенностью параллельной модели программирования является:

Ответ:

(1) нелокальный и динамический характер ошибок

(2) вероятность утраты детерминизма выполнения программы

(3) вероятность возникновения блокировок

Вопрос № 5-40010121

Особенностью последовательной модели программирования является:

Ответ:

(1) хорошая масштабируемость приложений

(2) плохая масштабируемость приложений

(3) универсальность

Вопрос № 6-40010122

Одной из схем организации параллельных программ является:

Ответ:

(1) схема "первый вошёл - первый вышел" (FIFO)

(2) иерархическая схема "хозяин-работник" (master-slave)

(3) схема "хозяин-работник" (master-slave)

Вопрос № 7-40010123

Потоки выполнения, относящиеся к одному параллельному приложению, характеризуются:

Ответ:

(1) общим адресным пространством

(2) общими дескрипторами файлов

(3) разными и непересекающимися адресными пространствами

Вопрос № 8-40010124

Процессы, относящиеся к одному параллельному приложению, характеризуются:

Ответ:

(1) общим адресным пространством

(2) разными и непересекающимися адресными пространствами

(3) в операционных системах UNIX одинаковым значением идентификатора PID (Process IDentifier)

Вопрос № 9-40010125

MPICH это:

Ответ:

(1) программный пакет, позволяющий разрабатывать параллельные программы на основе модели обмена сообщениями

(2) язык программирования, позволяющий разрабатывать многопоточные программы для вычислительных систем с общей памятью

(3) коммерческая библиотека, реализующая операции обмена сообщениями

Вопрос № 10-40010126

На программирование для систем с распределённой памятью ориентированы:

Ответ:

(1) POSIX Threads

(2) Intel (R) Cluster OpenMP

(3) OpenMP

Вопрос № 11-40010127

Одним из распространённых средств разработки многопоточных программ является:

Ответ:

(1) OpenMP

(2) PVM

(3) MPI

Вопрос № 12-40010128

Одним из распространённых средств разработки программ, основанных на модели обмена сообщениями, является:

Ответ:

(1) OpenMP

(2) POSIX Threads

(3) любая реализация MPI

Вопрос № 13-40010129

Компиляция MPI-программы выполняется командой:

Ответ:

(1) mpirun

(2) mpif90

(3) mpicxx

Вопрос № 14-40010130

Запуск MPI-программы выполняется командой:

Ответ:

(1) mpdboot

(2) mpirun

(3) mpiexec

Вопрос № 15-40010131

Запуск демонов mpd выполняется командой:

Ответ:

(1) mpdtrace

(2) mpdboot

(3) mpirun

Вопрос № 16-40010132

В MPI-программах на языке C устанавливается следующее соответствие между типами MPI и стандартными типами языка:

Ответ:

(1) MPI_FLOAT и float

(2) MPI_CHAR и signed char

(3) MPI_INT и unsigned int

Вопрос № 17-40010133

Получить значение ранга процесса можно с помощью подпрограммы MPI:

Ответ:

(1) MPI_Comm_size

(2) MPI_Comm_rank

(3) MPI_Init

Вопрос № 18-40010134

Ранг процесса:

Ответ:

(1) принимает как отрицательные, так и положительные целые значения

(2) назначается процессу системой

(3) принимает неотрицательные целые значения

Вопрос № 19-40010135

Коммуникатор это:

Ответ:

(1) область взаимодействия процессов, объединяющая все процессы с общим контекстом обмена

(2) устройство связи между вычислительными узлами кластера

(3) процедура MPI, выполняющая передачу сообщения от одного процесса другому

Вопрос № 20-40010136

MPI_COMM_NULL это:

Ответ:

(1) имя процедуры MPI

(2) имя стандартного "пустого" коммуникатора

(3) обозначение "нулевого" указателя в MPI

Вопрос № 21-40010137

MPI_COMM_WORLD это:

Ответ:

(1) имя стандартного коммуникатора, включающего все запущенные процессы MPI-программы

(2) имя стандартного коммуникатора, включающего все запущенные процессы, исполняющиеся в операционной системе

(3) имя процедуры MPI

Вопрос № 22-40010138

Первым по порядку вызовом подпрограммы MPI может быть вызов:

Ответ:

(1) MPI_Initialized

(2) MPI_Init

(3) MPI_Comm_size

Вопрос № 23-40010139

Последним по порядку вызовом подпрограммы MPI может быть вызов:

Ответ:

(1) MPI_Abort

(2) MPI_Finalize

(3) MPI_Finalized

Вопрос № 24-40010140

Отметьте правильную последовательность обращений к подпрограммам MPI:

Ответ:

(1) MPI_Comm_rank, MPI_Comm_size

(2) MPI_Comm_size, MPI_Comm_rank

(3) MPI_Comm_size, MPI_Init

Вопрос № 25-40010141

В двухточечном обмене сообщениями могут участвовать

Ответ:

(1) 3 процесса

(2) 2 процесса

(3) не менее двух процессов

Вопрос № 26-40010142

При двухточечном обмене:

Ответ:

(1) пересылаются данные одного типа

(2) принадлежность одному коммуникатору не имеет значения

(3) важна принадлежность одному коммуникатору

Вопрос № 27-40010143

При двухточечном обмене:

Ответ:

(1) допускается передача только одного скалярного значения

(2) количество вариантов отправки сообщений превосходит количество вариантов приёма

(3) за одну операцию обмена допускается передача структуры

Вопрос № 28-40010144

В MPI существуют следующие типы двухточечных обменов:

Ответ:

(1) блокирующие

(2) неблокирующие

(3) полублокирующие

Вопрос № 29-40010145

В MPI существуют следующие типы двухточечных обменов:

Ответ:

(1) обмен "по готовности"

(2) обмен с буферизацией

(3) быстрый обмен

Вопрос № 30-40010146

В MPI существуют следующие типы двухточечных обменов:

Ответ:

(1) стандартный обмен

(2) синхронный обмен

(3) обмен с буферизацией

Вопрос № 31-40010147

Стандартная блокирующая двухточечная передача выполняется подпрограммой

Ответ:

(1) MPI_Send

(2) MPI_Ssend

(3) MPI_Bsend

Вопрос № 32-40010148

Двухточечная передача с буферизацией выполняется подпрограммой

Ответ:

(1) MPI_Send

(2) MPI_Bsend

(3) MPI_Buffer_attach

Вопрос № 33-40010149

При организации двухточечного обмена с буферизацией используется подпрограмма:

Ответ:

(1) MPI_Bsend

(2) MPI_Buffer_detach

(3) MPI_Buffer_attach

Вопрос № 34-40010150

При организации двухточечного обмена с буферизацией размер буфера должен превосходить объём пересылаемых данных на величину:

Ответ:

(1) MPI_BSEND_OVERHEAD

(2) MPI_BUFFER_ATTACH

(3) 100 байт

Вопрос № 35-40010151

"Джокер" используется в подпрограмме двухточечного приема сообщения:

Ответ:

(1) для задания буфера приёма произвольного размера

(2) для обозначения произвольного источника сообщения

(3) для обозначения произвольного тега сообщения

Вопрос № 36-40010152

Пусть значение параметра count в подпрограмме приёма двухточечного сообщения больше, чем количество элементов в принятом сообщении. Тогда:

Ответ:

(1) выполнение программы завершится аварийно

(2) выполнение подпрограммы приёма завершится с кодом ошибки, но выполнение параллельной программы продолжится

(3) в буфере приёма изменится значение только тех элементов, которые соответствуют элементам фактически принятого сообщения

Вопрос № 37-40010153

При стандартной блокирующей двухточечной передаче сообщения:

Ответ:

(1) после завершения вызова можно использовать любые переменные, использовавшиеся в списке параметров

(2) выполнение параллельной программы приостанавливается до тех пор, пока сообщение будет принято процессом-адресатом

(3) после завершения вызова нельзя использовать переменные, использовавшиеся в списке параметров

Вопрос № 38-40010154

Укажите, является ли данное утверждение верным для буферизованного двухточечного обмена:

Ответ:

(1) при выполнении буферизованного обмена программист должен заранее создать буфер достаточного размера

(2) при выполнении буферизованного обмена к процессу можно подключить до 1024 буферов передачи

(3) после завершения работы с буфером его необходимо отключить

Вопрос № 39-40010155

Двухточечный обмен "по готовности" позволяет:

Ответ:

(1) упростить процесс отладки

(2) увеличить производительность параллельной MPI-программы

(3) повысить предсказуемость поведения параллельной программы

Вопрос № 40-40010156

При выполнении блокирующей передачи "по готовности":

Ответ:

(1) передача сообщения выполняется, как только будет готов результат вычислений, выполняемых процессом-источником сообщения

(2) передача должна начинаться до того, как зарегистрирован прием

(3) передача должна начинаться, если уже зарегистрирован соответствующий прием

Вопрос № 41-40010157

Операция совместных приёма и передачи:

Ответ:

(1) существует для двухточечных обменов

(2) не существует для двухточечных обменов

(3) существует только для коллективных обменов

Вопрос № 42-40010158

При неблокирующем двухточечном обмене:

Ответ:

(1) буфер передачи/приема можно использовать сразу после завершения вызова подпрограммы инициализации обмена

(2) передача сообщения происходит одновременно с выполнением процесса

(3) не блокируется выполнение всех процессов параллельной программы

Вопрос № 43-40010159
Неблокирующий вариант операций передачи сообщений существует для:

Ответ:

(1) стандартного обмена

(2) обмена с буферизацией

(3) обмена «по готовности»

Вопрос № 44-40010160

Неблокирующий прием сообщений реализован в MPI:

Ответ:

(1) одной подпрограммой

(2) двумя подпрограммами

(3) тремя подпрограммами

Вопрос № 45-40010161

Неблокирующая стандартная передача выполняется подпрограммой:

Ответ:

(1) MPI_Isend

(2) MPI_Issend

(3) MPI_Nonblocking_send

Вопрос № 46-40010162

Неблокирующий прием в MPI выполняется подпрограммой:

Ответ:

(1) MPI_Nonblocking_recv

(2) MPI_Irecv

(3) MPI_Recv

Вопрос № 47-40010163

Неблокирующая передача с буферизацией выполняется подпрограммой:

Ответ:

(1) MPI_Ibsend

(2) MPI_Bsend

(3) MPI_Immediate_bsend

Вопрос № 48-40010164

Первый этап выполнения неблокирующего обмена это:

Ответ:

(1) проверка доступности буфера обмена

(2) создание буфера обмена

(3) инициализация обмена

Вопрос № 49-40010165

Второй этап выполнения неблокирующего обмена это:

Ответ:

(1) создание буфера обмена

(2) проверка выполнения обмена

(3) проверка доступности буфера обмена

Вопрос № 50-40010166

Неблокирующий обмен позволяет:

Ответ:

(1) повысить производительность параллельной программы

(2) повысить предсказуемость поведения программы

(3) повысить надежность передачи сообщений

Вопрос № 51-40010167

Подпрограмма MPI_Wait предназначена для:

Ответ:

(1) блокирующей проверки выполнения обмена

(2) неблокирующей проверки выполнения обмена

(3) приостановки выполнения программы на заданный период времени

Вопрос № 52-40010168

Подпрограмма MPI_Test предназначена для:

Ответ:

(1) блокирующей проверки выполнения обмена

(2) неблокирующей проверки выполнения обмена

(3) проверки доступности коммуникационной среды

Вопрос № 53-40010169

После завершения вызова MPI_Wait:

Ответ:

(1) неблокирующий обмен выполнен

(2) неблокирующий обмен не выполнен

(3) возобновляется выполнение всех процессов, относящихся к данной параллельной программе

Вопрос № 54-40010170

Подпрограммы-пробники предназначены для:

Ответ:

(1) тестирования программ

(2) проверки фактической доставки сообщений

(3) тестирования коммуникационной подсистемы

Вопрос № 55-40010171

Размер полученного сообщения можно определить с помощью подпрограммы:

Ответ:

(1) MPI_Get_count

(2) MPI_Count

(3) MPI_Comm_size

Вопрос № 56-40010172

Подпрограмма MPI_Iprobe:

Ответ:

(1) является подпрограммой неблокирующей проверки сообщения

(2) является подпрограммой блокирующей проверки сообщения

(3) не является подпрограммой проверки сообщения

Вопрос № 57-40010173

Подпрограмма MPI_Testall выполняет проверку:

Ответ:

(1) завершения любого числа обменов

(2) завершения всех обменов

(3) завершения одного обмена

Вопрос № 58-40010174

Подпрограмма MPI_Testany выполняет проверку:

Ответ:

(1) завершения любого из нескольких обменов

(2) завершения всех обменов

(3) завершения первого обмена

Вопрос № 59-40010175

Подпрограмма MPI_Waitall выполняет проверку:

Ответ:

(1) завершения любого числа обменов

(2) завершения всех обменов

(3) завершения одного обмена

Вопрос № 60-40010176

В коллективном обмене принимают участие:

Ответ:

(1) два процесса

(2) все процессы из указанного коммуникатора

(3) произвольное количество процессов, но большее двух

Вопрос № 61-40010177

Операции коллективного обмена для инициировавших их процессов являются:

Ответ:

(1) блокирующими

(2) неблокирующими

(3) понятия "блокирующий/неблокирующий обмен" к ним неприменимы

Вопрос № 62-40010178

Для выполнения коллективного обмена:

Ответ:

(1) не требуется создание специального коммуникатора, если это не диктуется особенностями задачи

(2) требуется создание специального "коллективного" коммуникатора

(3) требуется инициализация посредством вызова подпрограммы MPI_Collective_init

Вопрос № 63-40010179

Широковещательная рассылка сообщения выполняется подпрограммой:

Ответ:

(1) MPI_Bcast

(2) MPI_Bsend

(3) MPI_Send

Вопрос № 64-40010180

Коллективная передача данных может сочетаться с двухточечным приемом:

Ответ:

(1) нет

(2) да

(3) иногда

Вопрос № 65-40010181

К числу коллективных обменов относятся:

Ответ:

(1) распределение данных

(2) сбор данных

(3) операция приведения

Вопрос № 66-40010182

К числу коллективных обменов не относится:

Ответ:

(1) обмен с буферизацией

(2) обмен по готовности

(3) векторная операция распределения данных

Вопрос № 67-40010183

Данная операция является операцией приведения:

Ответ:

(1) умножение двух матриц

(2) суммирование элементов массива

(3) определение максимального элемента

Вопрос № 68-40010184

Подпрограмма MPI_Bcast:

Ответ:

(1) пересылает всем остальным процессам разные фрагменты данных

(2) пересылает одну и ту же порцию данных всем остальным процессам

(3) выполняет операцию частичного приведения

Вопрос № 69-40010185

Подпрограмма MPI_Gather:

Ответ:

(1) выполняет сбор данных

(2) является операцией коллективного обмена

(3) выполняет широковещательную рассылку

Вопрос № 70-40010186

Подпрограмма MPI_Scatter:

Ответ:

(1) выполняет распределение данных

(2) выполняет широковещательную рассылку

(3) выполняет синхронизацию процессов

Вопрос № 71-40010187

Какие подпрограммы выполняют сборку данных?

Ответ:

(1) MPI_Gather

(2) MPI_Gatherv

(3) MPI_Allgather

Вопрос № 72-40010188

Подпрограмма выполняет сбор данных:

Ответ:

(1) MPI_Get_count

(2) MPI_Scatterv

(3) MPI_Allgather

Вопрос № 73-40010189

Подпрограмма является векторным вариантом операции коллективного обмена:

Ответ:

(1) MPI_Alltoallv

(2) MPI_Alltoall

(3) MPI_Allgatherv

Вопрос № 74-40010190

"Толщина барьера" при барьерной синхронизации это:

Ответ:

(1) количество процессов, в которых для продолжения выполнения должен быть выполнен вызов процедуры барьерной синхронизации

(2) величина, которая определяется пропускной способностью сети

(3) один из параметров при вызове подпрограммы барьерной синхронизации

Вопрос № 75-40010191

Подпрограмма MPI_Alltoall:

Ответ:

(1) выполняет передачу данных по схеме "каждый-всем"

(2) выполняет операцию сканирования (частичной редукции)

(3) выполняет операцию полной редукции

Вопрос № 76-40010192

Подпрограмма MPI_Scan:

Ответ:

(1) выполняет операцию полной редукции

(2) выполняет проверку доступности узлов многопроцессорной вычислительной системы

(3) выполняет операцию частичной редукции

Экзамен

При составлении экзаменационных билетов рекомендуется сочетать теоретические вопросы и практические задания таким образом, чтобы обеспечить их одинаковое соотношение в билете и исключить дублирование тематики. К экзамену допускаются студенты, успешно выполнившие курс лабораторных, практических работ, защитившие курсовой проект.

Тематика теоретического экзаменационного материала

I Тематический раздел «Организация компьютерных систем»

Понятие архитектуры ЭВМ.
Организация, классификация, характеристики процессоров
Запоминающие устройства ЭВМ
Организация, классификация, характеристики интерфейсов ЭВМ
Эволюция архитектуры x86. Возникновение и развитие процессоров семейства x86. Технические особенности поколений модели Core i7.
Архитектура IA-32. Микроархитектура P6. Микроархитектура NetBurst.
Программная модель IA-32.
Организация памяти.
Организация ЭВМ и язык ассемблера. Общие сведения. Основные конструкции. Команды пересылки данных. Работа с адресами и указателями. Работа со стеком. Арифметические команды. Команды сложения-вычитания. Команды умножения-деления. Команды расширения знака. Программирование ветвящихся вычислительных процессов. Программирование циклических вычислительных процессов. Режимы адресации. Цепочечные команды.
Принципы построения параллельных вычислительных систем. Пути достижения параллелизма. Классификация вычислительных систем. Характеристика типовых схем коммуникации в многопроцессорных вычислительных системах.

II Тематический раздел «Организация компьютерных сетей»

История развития компьютерных сетей. Многотерминальные системы – прообраз сети. Появление глобальных сетей. Первые локальные сети. Создание стандартных технологий локальных сетей. Современные тенденции развития компьютерных сетей.
Основные понятия и определения. Классификации компьютерных сетей. Основные программные и аппаратные компоненты сети.
Модель OSI (Эталонная модель взаимодействия открытых систем). Основные принципы. Протоколы верхних уровней. Транспортная служба модели OSI. Стек протоколов нижних уровней модели OSI.
Методы коммутации информации.

Способы организации виртуальных каналов и управления потоками данных.

Методы маршрутизации информации.

Принципы межсетевого взаимодействия. Межсетевое взаимодействие посредством протокола X.75. Подход соединения сетей в рамках архитектуры протоколов DARPA. Межсетевой протокол IPv4. Протокол ICMP. IPv4-адресация. Маршрутизация IP-дейтаграмм. Пример подключения локальной сети организации к Интернет.

Транспортные протоколы. Базовые характеристики транспортных протоколов. TCP, UDP. Протокол управления передачей TCP. Протокол UDP.

Базовые технологии локальных сетей. Протоколы и стандарты локальных сетей. Структура стандартов IEEE 802.x. Технология Ethernet (802.3). MAC-адреса. Форматы кадров технологии Ethernet. Метод доступа CSMA/CD. Максимальная производительность сети Ethernet. Физические стандарты 10M Ethernet. Коммутируемые сети Ethernet. Логическая структуризация сетей и мосты. Алгоритм прозрачного моста IEEE 802.1D. Коммутаторы. Параллельная коммутация. Дуплексный режим работы. Неблокирующие коммутаторы. Борьба с перегрузками. Скоростные версии Ethernet. Физические уровни технологии Fast Ethernet. Gigabit Ethernet. 10G Ethernet. 100G и 40G Ethernet. Беспроводные локальные сети IEEE 802.11. Проблемы и области применения беспроводных локальных сетей. Стандарт 802.11. Физические уровни стандарта 802.11.

Первичные сети. Назначение и типы первичных сетей. Сети PDH. Сети SONET/SDH. Типы оборудования сети SDH. Типовые топологии сетей SDH.

Тематика практического экзаменационного материала

Практический экзаменационный материал базируется на изученном студентами курсе практических занятий и может включать следующие направления практических заданий:

- по тематическому разделу «Организация компьютерных систем» – расчет параметров компьютерных систем для различных дисциплин обслуживания заявок; расчет критериев эффективности цифровых управляющих систем; оценка трудоемкости вычислительного алгоритма, реализуемого компьютерной системой; расчет параметров вычислительного конвейера;

- по тематическому разделу «Организация компьютерных сетей» – задачи маршрутизации сетевого трафика; IP-адресация; структурирование IP-сети с использованием масок; расчет характеристик коммутаторов локальных вычислительных сетей.

4 ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

4.1 Учебная программа

Учреждение образования
«Брестский государственный технический университет»

УТВЕРЖДАЮ
Первый проректор

_____ М.В. Нерода
_____ 2022 г.

Регистрационный № УД-22-1-171/уч.

Компьютерные системы и сети

Учебная программа учреждения высшего образования по учебной дисциплине для специальности:
1-40 01 01 Программное обеспечение информационных технологий

2022

Учебная программа составлена на основе образовательного стандарта ОСВО 1–40 01 01–2021 и учебного плана специальности 1-40 01 01 «Программное обеспечение информационных технологий».

СОСТАВИТЕЛЬ:
Савицкий Ю.В., доцент кафедры интеллектуальных информационных технологий, кандидат технических наук, доцент

РЕЦЕНЗЕНТЫ:
Коренкович О.Г., начальник отдела программирования ОАО «Савушкин продукт»
Костюк Д.А., доцент кафедры ЭВМ и систем учреждения образования «Брестский государственный технический университет», кандидат технических наук, доцент

РЕКОМЕНДОВАНА К УТВЕРЖДЕНИЮ:
Кафедрой интеллектуальных информационных технологий
Заведующий кафедрой В.А. Головко
(протокол №_9_ от _17.06.2.22___);

Методической комиссией факультета электронно-информационных систем
Председатель методической комиссии С.С. Дереченник
(протокол № _10_ от _25.06.2022_);

Научно-методическим советом БрГТУ (протокол № _7_ от _29.06.__2022)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Место учебной дисциплины.

Дисциплина «Компьютерные системы и сети» является одной из базовых в процессе подготовки студентов по специальности «Программное обеспечение информационных технологий»; относится к государственному компоненту; в значительной степени носит системный характер, формируя основу для освоения ряда последующих курсов, организации дипломного проектирования; обеспечивает выработку систематизированных знаний в области по архитектуре, логической организации, принципам построения и функционирования современных ЭВМ и сетей.

Цель преподавания учебной дисциплины:

- изучение основ организации, принципов построения и функционирования современных вычислительных комплексов, систем и сетей различного назначения; методам анализа их характеристик. Формирование у студентов систематизированного представления о принципах построения систем параллельной обработки информации. Получение практических навыков машинно-ориентированного и сетевого программирования.

Задачи учебной дисциплины:

- приобретение знаний по организации архитектур современных компьютерных систем и принципам их функционирования;
- изучение принципов построения современных компьютерных сетей, сетевых протоколов, сетевого программного обеспечения;
- овладение методами машинно-ориентированного и сетевого программирования.

Изучение учебной дисциплины должно обеспечить формирование у студента следующих компетенций:

универсальных (УК):

- УК-1. Владеть основами исследовательской деятельности, осуществлять поиск, анализ и синтез информации;
- УК-5. Обладать навыками саморазвития и совершенствования в профессиональной деятельности;
- УК-6. Проявлять инициативу и адаптироваться к изменениям в профессиональной деятельности;

базовых профессиональных (БПК):

- БПК-15. Использовать общепринятые подходы в построении, конфигурировании и администрировании компьютерных систем и сетей.

В результате изучения учебной дисциплины студент должен:

- знать: базовые понятия языка ассемблера, архитектуру компьютеров на основе процессоров семейства Intel, основные аспекты современного программирования на ассемблере; основные виды архитектур глобальных и локальных вычислительных сетей, способы логической и физической организации; принципы многоуровневой организации архитектуры взаимодействия открытых систем (ВОС); назначение, представляемую службу и функции отдельных уровней модели ВОС; основные понятия, принципы

построения и иерархию современных сетевых протоколов; основы международной стандартизации сетей; архитектуры, принципы организации и проектирования современных локальных вычислительных сетей;

- уметь: реализовывать алгоритмы на языке ассемблера; по техническим требованиям сформировать логическую и физическую структуру сети, определить режимы ее функционирования; ставить и решать задачи эффективного применения вычислительных архитектур в сетевых приложениях;

- приобрести навыки: написания и отладки программ на языке ассемблера, внедрения ассемблера в программы на языках высокого уровня; разработки и использования сетевого программного обеспечения;

- иметь представление: о возможностях применения языка ассемблера для оптимизации программ; о перспективах развития вычислительных сетей; о направлениях научно-исследовательских работ в области создания, совершенствования их архитектур, технических средств, программного обеспечения.

Связи с другими учебными дисциплинами. Дисциплина базируется на знаниях, полученных студентами при изучении дисциплин «Основы алгоритмизации и программирования», «Алгоритмы и структуры данных», «Операционные системы» и др.

План учебной дисциплины для дневной формы получения высшего образования

Код специальности	Наименование специальности	Курс	Семестр	Всего учебных часов	Количество зачетных единиц	Аудиторных часов (в соответствии с учебным планом УВО)					Академических часов на курсовой проект (работу)	Форма текущей аттестации
						Всего	Лекции	Лабораторные занятия	Практические занятия	Семинары		
1-40 01 01	«Программное обеспечение информационных технологий»	2	4	196	5	88	32	40	16	-	40	письмен-ный экзамен

1. СОДЕРЖАНИЕ УЧЕБНОГО МАТЕРИАЛА

1.1. ЛЕКЦИОННЫЕ ЗАНЯТИЯ, ИХ СОДЕРЖАНИЕ

РАЗДЕЛ 1. НАЧАЛЬНЫЕ СВЕДЕНИЯ ОБ ОРГАНИЗАЦИИ КОМПЬЮТЕРНЫХ СИСТЕМ И СЕТЕЙ

Тема 1. Введение. Основные понятия

Предмет, цели, задачи курса. История и перспективы развития компьютерных систем и сетей, их базовых компонентов.

РАЗДЕЛ 2. АРХИТЕКТУРА ПЕРСОНАЛЬНЫХ ЭВМ

Тема 1. Принципы организации вычислительных систем

Фундаментальные принципы организации вычислительных систем. Программное управление вычислительным процессом.

Общее понятие об архитектуре вычислительной системы: архитектура памяти, архитектура ввода-вывода, архитектура процессора. Архитектура и свойства машины фон Неймана. Структурная схема персонального компьютера (на типовом примере Pentium PRO I II II). Машинный язык и язык ассемблера.

Тема 2. Организация, классификация, характеристики процессоров

Определение процессора. Основные характеристики процессора – технологический процесс, тактовая частота и производительность, потребляемая мощность. Типы архитектур процессора: RISC- и CISC-процессоры, Гарвардская и Принстонская архитектура, суперскалярные и EPIC-процессоры, процессоры с фиксированной и модифицируемой системой команд. Специализированные процессоры и их функциональные характеристики: математический, физический, сигнальный, звуковой, графический, сетевой, криптографический процессоры.

Тема 3. Запоминающие устройства ЭВМ

Классификация запоминающих устройств ЭВМ. Внешняя память – типы, организация, основные характеристики. Внутренняя память ROM, RAM. Особенности организации DRAM, SRAM, SDRAM, RDRAM, DDR. Поколения DDR и их характеристики. Особенности организации многоканальной памяти DDR – DDR II, DDR III.

Основные количественные характеристики ОЗУ. Понятие "бутылочного горлышка" (bottle neck) фон-неймановской архитектуры. Способы увеличения производительности памяти. Многоуровневая организация оперативной памяти, понятие кэш-памяти. Классификация кэш-памяти по конструктивному исполнению, по уровням.

Тема 4. Организация, классификация, характеристики интерфейсов ЭВМ

Классификация интерфейса с устройствами ввода-вывода: по пересечению с

адресным пространством памяти (основным адресным пространством) процессора: по способу синхронизации работы процессора с устройствами ввода-вывода, по количеству линий данных, по направлению передачи, по топологии. Пропускная способность интерфейса и методика ее определения.

Интерфейс ввода-вывода IDE (ATA), ATAPI, SATA – назначение, отличительные особенности каждой модификации, пропускная способность.

Интерфейс ввода-вывода SCSI (Small Computer System Interface) – назначение, отличительные особенности каждой из модификаций, пропускная способность. Особенности и параметры интерфейса Serial Attached SCSI (SAS).

Интерфейс ввода-вывода USB. Назначение и организация. Характеристики пропускной способности USB 1.x, USB 2.0, USB 3.x, USB4, Inter-Chip USB, Wireless USB.

Тема 5. Архитектура IA-32

Программная модель архитектуры IA-32. Режимы работы процессора архитектуры IA-32.

Набор регистров процессора архитектуры IA-32. Назначение основных флагов регистра состояния.

Организация памяти компьютера архитектуры IA-32. Сегментированная модель памяти. Формирование физического адреса в архитектуре IA-32 в реальном режиме и в защищенном режиме.

Формат машинных команд IA-32. Функциональная классификация машинных команд IA-32.

Тема 6. Организация ассемблера

Синтаксис ассемблера. Структура программы на ассемблере. Директивы сегментации.

Переменные и директивы их описания. Стандартные типы данных и их размещение в памяти.

Способы адресации операндов в ассемблере.

Команды передачи данных. Арифметические команды сложения/вычитания/сравнения. Арифметические команды умножения/деления.

Организация условного/безусловного перехода. Команды условного/безусловного перехода.

Организация циклов при помощи команды LOOP, организация вложенных циклов.

Организация подпрограмм. Команды для организации подпрограмм.

Организация системы прерываний. Типы прерываний в компьютерных системах. Команды прерываний.

РАЗДЕЛ 3. ПАРАЛЛЕЛЬНЫЕ КОМПЬЮТЕРНЫЕ СИСТЕМЫ

Тема 1. Параллельные вычисления

Понятие параллельных вычислений. Требования к архитектурным принципам построения компьютерной системы для реализации параллельных

вычислений. режимы выполнения независимых частей программы при организации параллельных вычислений.

Тема 2. Классификация Флинна

Классификация вычислительных систем по систематике Флинна и пути достижения параллелизма обработки для каждой из архитектур классификации.

Тема 3. Варианты организации параллельных вычислительных систем

Варианты организации систем MIMD класса «мультипроцессоры». Особенности архитектур NUMA (NCC-NUMA, CC-NUMA, COMA). Примеры архитектур (реализации Sun, IBM, Cray и др.).

Варианты организации систем MIMD класса «мультикомпьютеры» (NORMA). Понятия транспьютерной системы, кластерной системы. Особенности построения архитектур MPP, Clusters. Примеры архитектур (реализации IBM, Intel и др.).

РАЗДЕЛ 4. ОБЩИЕ ПРИНЦИПЫ ОРГАНИЗАЦИИ КОМПЬЮТЕРНЫХ СЕТЕЙ

Тема 1. Организация компьютерных сетей

Эволюция сетей ЭВМ. Обобщенная (функционально-логическая и физическая) структура вычислительной сети и назначение основных компонентов.

Тема 2. Эталонная модель взаимодействия открытых систем

Эталонная модель взаимодействия открытых систем (ЭМВОС): основные понятия и определения; особенности организации; назначения и функции отдельных уровней ЭМВОС.

Тема 3. Организация базовой сети обмена данными

Способы организации обмена данными в вычислительной сети. Методы коммутации информации: коммутация каналов и пакетов, смешанная и интегральная коммутация. Способы организации виртуальных каналов и управления потоками данных: протоколы с остановками и ожиданиями, с N-возвращениями, с селективным повторением. Использование метода окна в процедурах управления потоками данных.

Тема 4. Методы и протоколы маршрутизации

Методы маршрутизации информации в компьютерных сетях: простая, фиксированная, адаптивная. Протоколы RIP, OSPF.

Тема 5. Принципы межсетевого взаимодействия

Принципы межсетевого взаимодействия, назначение и типы межсетевых

шлюзов. Межсетевое взаимодействие для протоколов без установления логического соединения с применением межсетевых дейтаграмм. Межсетевой протокол IPv4: назначение, структура заголовка, выполняемые функции. Протокол IPv6 – основные отличия.

Тема 6. IP-адресация

Типы IP-адресов и классы подсетей. Протокол ICMP. Понятие маски. Использование масок для структуризации IP сетей. Перспективы IP-адресации.

Тема 7. Транспортная служба

Классы транспортных протоколов и типы сетевых соединений. Функции транспортной службы. Организация транспортного протокола TCP, формат заголовка. Процедура передачи данных и метод окна в TCP. Адаптивные свойства протокола TCP. Особенности и сфера протокола UDP.

Тема 8. Протоколы высоких уровней

Задачи и типы протоколов высоких уровней. Протоколы прикладного уровня: протокол виртуального терминала VTP; протоколы передачи, доступа и управления файлами FTAM; протоколы управления удаленными заданиями; протоколы управления и пересылки сообщений X.400 и справочной службы имен X.500. Протоколы административного управления сетью.

Тема 9. Организация локальных вычислительных сетей

Особенности организации модели взаимодействия для локальных вычислительных сетей (ЛВС). Протоколы и стандарты ЛВС. Спецификации протоколов LLC. Назначение и функции протокола, типы кадров

Тема 10. Технологии семейства Ethernet

Технологии семейства Ethernet (IEEE 802.3x). Иерархия стандартов Ethernet. Метод доступа CSMA/CD. Форматы кадров Ethernet.

Тема 11. Основные характеристики базовых стандартов IEEE 802.3x

Основные характеристики стандартов серии 10Base-x, 100Base-x. Основные характеристики стандартов серии 1000Base-x. Технология 10GE.

Тема 12. Логическая структуризация ЛВС

Домен коллизий и логическая структуризация сетей семейства Ethernet. Принцип функционирования коммутаторов локальных сетей (на примере коммутатора EtherSwitch). Построение локальных сетей на базе коммутаторов.

Тема 13. Беспроводные технологии ЛВС

Технологии беспроводных ЛВС семейства стандарта IEEE 802.11. Характеристика методов множественного доступа к каналу. Особенности конфигурации беспроводных сетей.

Тема 14. Цифровые первичные сети

Общие сведения о цифровых первичных сетях. Технология PDH. Технология SDH. Примеры архитектур и характеристики высокоскоростных технологий глобальных вычислительных сетей.

1.2. ПРАКТИЧЕСКИЕ (СЕМИНАРСКИЕ) ЗАНЯТИЯ, ИХ СОДЕРЖАНИЕ

1. Дисциплины обслуживания в компьютерных системах.
2. Критерии эффективности компьютерных систем и их элементов.
3. Оценка трудоемкости вычислительного алгоритма.
4. Расчет параметров вычислительного конвейера.
5. Алгоритмы маршрутизации сетевой информации.
6. IP-адресация.
7. Структурирование IP-сетей и использованием масок.
8. Изучение характеристик коммутаторов ЛВС.

1.3. ПЕРЕЧЕНЬ ТЕМ ЛАБОРАТОРНЫХ ЗАНЯТИЙ, ИХ НАЗВАНИЕ

1. Изучение аппаратно-программной архитектуры процессоров семейства Intel: программирование на языке Ассемблера – разработка ассемблерной вставки.
2. Изучение аппаратно-программной архитектуры процессоров семейства Intel: программирование на языке Ассемблера – обработка символьных данных.
3. Изучение аппаратно-программной архитектуры процессоров семейства Intel: программирование на языке Ассемблера – макроопределения.
4. Архитектура и программирование сопроцессора. Использование целочисленных команд.
5. Архитектура и программирование сопроцессора. Использование вещественных команд.
6. Изучение пакета Cisco Packet Tracer. Начальная конфигурация маршрутизатора Cisco.
7. Настройка статической маршрутизации на устройствах Cisco. Настройка маршрутизации по умолчанию на устройствах Cisco.
8. Настройка динамической маршрутизации с помощью протокола RIP на устройствах Cisco.

2. ТРЕБОВАНИЯ К КУРСОВОМУ ПРОЕКТУ (РАБОТЕ)

Целью курсового проекта является закрепление и углубление теоретических знаний, формирование практических умений и навыков по следующим направлениям:

- организация высокопроизводительных компьютерных систем;
- концепции параллельного программирования;

- технологии построения систем параллельной обработки на базе кластерных / мультипроцессорных систем и компьютерных сетей.

Количество часов, отведенных учебным планом по специальности 1-40 01 01 «Программное обеспечение информационных технологий» на курсовое проектирование – 40.

Пояснительная записка по курсовому проекту может включать следующие разделы:

1. Анализ задач проектирования (согласно выданному заданию).
2. Разработка вариантов схем параллелизации, их оценка. Обоснование выбора варианта.
3. Разработка программных модулей систем параллельной и последовательной обработки.
4. Тестирование и сравнительный анализ производительности параллельного и последовательного приложений.

Примерный перечень тем курсовых проектов (работ)

1. Реализация параллельной обработки для кластерных / мультипроцессорных систем.
2. Проектирование распределенных систем обработки данных.
3. Разработка сетевого программного обеспечения на базе сетевых протоколов и платформ.

3. УЧЕБНО-МЕТОДИЧЕСКАЯ ЧАСТЬ

3.1. УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА УЧЕБНОЙ ДИСЦИПЛИНЫ
для дневной формы получения высшего образования

Номер раздела, темы, лекции	Название раздела, темы	Количество аудиторных часов				Количество часов самост. работы	Форма контроля знаний
		Лекции	Лабораторные занятия	Практические занятия	Семинарские занятия		
1	2	3	4	5	6	7	8
1	РАЗДЕЛ 1. НАЧАЛЬНЫЕ СВЕДЕНИЯ ОБ ОРГАНИЗАЦИИ КОМПЬЮТЕРНЫХ СИСТЕМ И СЕТЕЙ						
1.1.1	Тема 1. Введение. Основные понятия	2				4	контрольные опросы
2	РАЗДЕЛ 2. АРХИТЕКТУРА ПЕРСОНАЛЬНЫХ ЭВМ						
2.1.2	Тема 1. Принципы организации вычислительных систем	1				4	контрольные опросы
2.2.2	Тема 2. Организация, классификация, характеристики процессоров	1				6	контрольные опросы
	Практ. зан. №1. Дисциплины обслуживания в компьютерных системах			2			письменные отчеты
2.3.3	Тема 3. Запоминающие устройства ЭВМ	1				4	письменные отчеты
2.4.3	Тема 4. Организация, классификация, характеристики интерфейсов ЭВМ	1				4	контрольные опросы
2.5.4	Тема 5. Архитектура IA-32	1				4	контрольные опросы
	Практ. зан. №2. Критерии эффективности компьютерных систем и их элементов			2			письменные отчеты
2.6	Тема 6. Организация ассемблера						
2.6.4	Синтаксис ассемблера. Структура программы на ассемблере. Директивы сегментации. Переменные и директивы их описания. Стандартные типы данных и их размещение в памяти. Способы адресации операндов в ассемблере.	1				6	контрольные опросы
	Лаб. раб. №1. Изучение аппаратно-программной архитектуры процессоров семейства Intel: программирование на языке Ассемблера - разработка ассемблерной вставки		6				письменные отчеты
2.6.5	Команды передачи данных. Арифметические команды сложения/вычитания/сравнения. Арифметические команды умножения/деления. Организация и команды условного/безусловного перехода.	2				4	контрольные опросы

1	2	3	4	5	6	7	8
	Практ. зан. №3. Оценка трудоемкости вычислительного алгоритма			2			письменные отчеты
2.6.6	Организация циклов при помощи команды LOOP, организация вложенных циклов. Организация подпрограмм. Команды для организации подпрограмм. Организация системы прерываний. Типы прерываний в компьютерных системах. Команды прерываний.	2				4	контрольные опросы
	Лаб. раб. №2. Изучение аппаратно-программной архитектуры процессоров семейства Intel: программирование на языке Ассемблера - обработка символьных данных		6				письменные отчеты
3	РАЗДЕЛ 3. ПАРАЛЛЕЛЬНЫЕ КОМПЬЮТЕРНЫЕ СИСТЕМЫ						
3.1.7	Тема 1. Параллельные вычисления	1				4	контрольные опросы
	Практ. зан. №4. Расчет параметров вычислительного конвейера			2			письменные отчеты
3.2.7	Тема 2. Классификация Флинна	1				4	контрольные опросы
	Лаб. раб. №3. Изучение аппаратно-программной архитектуры процессоров семейства Intel: программирование на языке Ассемблера - макроопределения		4				письменные отчеты
3.3.8	Тема 3. Варианты организации параллельных вычислительных систем	2				4	контрольные опросы
4	РАЗДЕЛ 4. ОБЩИЕ ПРИНЦИПЫ ОРГАНИЗАЦИИ КОМПЬЮТЕРНЫХ СЕТЕЙ						контрольные опросы
4.1.9	Тема 1. Организация компьютерных сетей	1				4	контрольные опросы
	Лаб. раб. №4. Архитектура и программирование сопроцессора. Использование целочисленных команд		4				письменные отчеты
4.2.9	Тема 2. Эталонная модель взаимодействия открытых систем	1				4	контрольные опросы
4.3.10	Тема 3. Организация базовой сети обмена данными	1				4	контрольные опросы
4.4.10	Тема 4. Методы и протоколы маршрутизации	1				4	контрольные опросы
	Практ. зан. №5. Алгоритмы маршрутизации сетевой информации.			2			письменные отчеты
4.5.11	Тема 5. Принципы межсетевого взаимодействия	1				4	контрольные опросы
	Лаб. раб. №5. Архитектура и программирование сопроцессора. Использование вещественных команд		4				письменные отчеты
4.6.11	Тема 6. IP-адресация	1				4	контрольные опросы
	Практ. зан. №6. IP-адресация.			2			письменные отчеты
4.7.12	Тема 7. Транспортная служба	2				4	контрольные опросы
4.8.13	Тема 8. Протоколы высоких уровней	1				4	контрольные опросы

1	2	3	4	5	6	7	8
	Лаб. раб. №6. Изучение пакета Cisco Packet Tracer. Начальная конфигурация маршрутизатора Cisco		4				письменные отчеты
	Практ. зан. №7. Структурирование IP-сетей и использованием масок			2			письменные отчеты
4.9.13	Тема 9. Организация локальных вычислительных сетей	1				4	контрольные опросы
4.10.14	Тема 10. Технологии семейства Ethernet	2				4	контрольные опросы
	Лаб. раб. №7. Настройка статической маршрутизации на устройствах Cisco. Настройка маршрутизации по умолчанию на устройствах Cisco		6				письменные отчеты
4.11.15	Тема 11. Основные характеристики базовых стандартов IEEE 802.3x	1				4	контрольные опросы
4.12.15	Тема 12. Логическая структуризация ЛВС	1				4	контрольные опросы
	Практ. зан. №8. Изучение характеристик коммутаторов ЛВС			2			письменные отчеты
	Лаб. раб. №8. Настройка динамической маршрутизации с помощью протокола RIP на устройствах Cisco		6				письменные отчеты
4.13.16	Тема 13. Беспроводные технологии ЛВС	1				4	контрольные опросы
4.14.16	Тема 14. Цифровые первичные сети	1				4	контрольные опросы

3. ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

3.1. Перечень литературы

Основная

1. С. А. Орлов. Организация ЭВМ и систем: Учебник для ВУЗов. Для бакалавров и магистров. 4-е издание. – СПб.: Питер, 2020. – 688 с.

2. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для ВУЗов. Юбилейное издание. – СПб.: Питер, 2020. – 1008 с.

3. Таненбаум Э. Архитектура компьютера / Э. Таненбаум, Т. Остин. – 6-е изд. – Санкт-Петербург: Питер, 2019. – 816 с.

4. Юров В. И. Assembler: Учебник / В. И. Юров. – 2-е изд. – Санкт-Петербург: Питер, 2011. – 637 с.

Дополнительная

1. Хамахер, К. Организация ЭВМ / К. Хамахер, З. Вранешич, С. Заки. – 5-е изд. – СПб. : Питер, 2003. – 848 с.

2. Таненбаум Э. Архитектура компьютера / Э. Таненбаум, Д. Уэзеролл. – 5-е изд. – Санкт-Петербург: Питер, 2019. – 960 с.

3. Пешков А.Т. Организация и функционирование ЭВМ: Методическое пособие: В 3 ч. Ч. 1: Арифметические основы ЭВМ / А.Т. Пешков. – Мн.: БГУИР, 2004. – 61 с.

4. Пешков А.Т. Организация и функционирование ЭВМ: Методическое пособие: В 3 ч. Ч. 2: Логические основы ЭВМ / А.Т. Пешков. – Мн.: БГУИР, 2005. – 36 с.

5. Пешков А.Т., Кобайло А.С. Организация и функционирование ЭВМ: Методическое пособие: В 3 ч. Ч. 3: Схемотехнические основы ЭВМ / А.Т. Пешков, А. С. Кобайло. – Мн.: БГУИР, 2009. – 70 с.

3.2. Перечень средств диагностики результатов учебной деятельности

- контрольные опросы;
- письменные отчеты по лабораторным работам;
- письменный экзамен.

3.4. Методические рекомендации по организации и выполнению самостоятельной работы

Самостоятельная работа предполагает:

- ведение и систематическую проработку конспекта лекций и учебной литературы;
- подготовку к лабораторным занятиям – включает проработку соответствующих разделов рекомендованной литературы и конспекта лекций по тематике лабораторных работ;

- подготовку письменных отчетов по результатам выполнения индивидуальных заданий лабораторных работ;

- подготовку к письменному экзамену по дисциплине.

Ссылки на рекомендуемые источники, учебную литературу (в разрезе тем изучаемой дисциплины) представлены в таблице ниже.

Тема учебной дисциплины	Литература
РАЗДЕЛ 1. НАЧАЛЬНЫЕ СВЕДЕНИЯ ОБ ОРГАНИЗАЦИИ КОМПЬЮТЕРНЫХ СИСТЕМ И СЕТЕЙ	
Тема 1. Введение. Основные понятия	[1, Введение, §1], [2, §1], [3, §1], [4, §1]
РАЗДЕЛ 2. АРХИТЕКТУРА ПЕРСОНАЛЬНЫХ ЭВМ	
Тема 1. Принципы организации вычислительных систем	[1, §3], [3, §2]
Тема 2. Организация, классификация, характеристики процессоров	[1, §9], [3, §2]
Тема 3. Запоминающие устройства ЭВМ	[1, §6], [3, §2]
Тема 4. Организация, классификация, характеристики интерфейсов ЭВМ	[1, §7], [3, §2]
Тема 5. Архитектура IA-32	[4, §2]
Тема 6. Организация ассемблера	[4, §3-§14]
РАЗДЕЛ 3. ПАРАЛЛЕЛЬНЫЕ КОМПЬЮТЕРНЫЕ СИСТЕМЫ	
Тема 1. Параллельные вычисления	[1, §10], [3, §8]
Тема 2. Классификация Флинна	[1, §10-§12]
Тема 3. Варианты организации параллельных вычислительных систем	[1, §13-§14]
РАЗДЕЛ 4. ОБЩИЕ ПРИНЦИПЫ ОРГАНИЗАЦИИ КОМПЬЮТЕРНЫХ СЕТЕЙ	
Тема 1. Организация компьютерных сетей	[2, §1-§2]
Тема 2. Эталонная модель взаимодействия открытых систем	[2, §4]
Тема 3. Организация базовой сети обмена данными	[2, §2-§3]
Тема 4. Методы и протоколы маршрутизации	[2, §14, §16, §17]
Тема 5. Принципы межсетевого взаимодействия	[2, §14]
Тема 6. IP-адресация	[2, §14]
Тема 7. Транспортная служба	[2, §15]
Тема 8. Протоколы высоких уровней	[2, §24, §25]
Тема 9. Организация локальных вычислительных сетей	[2, §10]
Тема 10. Технологии семейства Ethernet	[2, §10]
Тема 11. Основные характеристики базовых стандартов IEEE 802.3x	[2, §10]
Тема 12. Логическая структуризация ЛВС	[2, §10, §11]
Тема 13. Беспроводные технологии ЛВС	[2, §21, §22]
Тема 14. Цифровые первичные сети	[2, §19, §20]