МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Е.А. Мыцко, С.А. Андреев, А.Д. Чередов

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ

*Рекомендовано в качестве учебного пособия
Редакционно-издательским советом
Томского политехнического университета*

5-е издание, переработанное и дополненное

Издательство
Томского политехнического университета
2018

УДК 004.38+004.7(075.8)
ББК 32.973+32.973.202я73
Ч–462

Мыцко Е.А., Андреев С.А., А.Д. Чередов

Ч–462 Организация ЭВМ и систем: учебное пособие / Е.А. Мыцко, С.А. Андреев; А.Д. Чередов Томский политехнический университет. – 5-е изд., перераб. и доп. – Томск: Изд-во Томского политехнического университета, 2018. – 296 с.

В учебном пособии рассматриваются основные вопросы, связанные с организацией ЭВМ и систем: архитектуры, характеристики и классификация ЭВМ; функциональная и структурная организация ЭВМ и центрального процессора; принципы организации подсистемы памяти ЭВМ и вычислительных систем; принципы организации подсистемы ввода-вывода; функциональная и структурная организация графического процессора, архитектуры и способы организации многопроцессорных вычислительных систем.

Учебное пособие подготовлено на отделении информационных технологий ИШИТР ТПУ и предназначено для студентов, обучающихся по направлению 09.03.01 «Информатика и вычислительная техника».

УДК 004.38+004.7(075.8)
ББК 32.973+32.973.202я73

Рецензенты

Кандидат технических наук,
зам. начальник управления информационных технологий
ОАО «Востокгазпром»
П.М. Острасть

Кандидат технических наук,
доцент кафедры программирования ТГУ
С.А. Останин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В последнее десятилетие в России бурно осуществляется информатизация и компьютеризация всех сфер человеческой деятельности. Компьютеры или электронные вычислительные машины (ЭВМ), оснащенные специальным программным обеспечением, являются технической базой и инструментом для вычислительных, информационных и автоматизированных систем.

При изучении дисциплины «Организация ЭВМ» в процессе подготовки бакалавров по направлению «Информатика и вычислительная техника» студенты используют знания, умения и навыки, полученные по дисциплинам «Информатика», «Дискретная математика», «Основы теории передачи информации», «Введение в информационные технологии», которые являются пререквизитами данной дисциплины.

Целью дисциплины «Организация ЭВМ» является освоение теоретических основ функциональной и структурной организации ЭВМ, включающих архитектуры ЭВМ, технические характеристики, классификации, особенности организации различных типов ЭВМ и ее составных частей (процессора, памяти, ввода-вывода), а также современное состояние и тенденции развития средств вычислительной техники.

В результате изучения этой дисциплины студент должен уметь применять полученные знания для решения практических задач: проводить анализ всего многообразия типов ЭВМ, осуществлять анализ параметров основных технических средств ЭВМ, выбирать и тестировать аппаратные средства вычислительных систем. Он должен владеть навыками конфигурирования компьютеров различного назначения.

Знания, умения и навыки, полученные при изучении дисциплины «Организация ЭВМ», необходимы для освоения ряда других дисциплин: «Операционные системы», «Периферийные устройства», «Сети и телекоммуникации» и др.

В процессе освоения дисциплины у студентов формируются компетенции, дающие им возможность разрабатывать технические задания на оснащение отделов, лабораторий, офисов компьютерным оборудованием, осуществлять наладку и обслуживание этого оборудования.

В предлагаемом вниманию читателя учебном пособии рассматриваются современные проблемы, связанные с функциональной и структурной организацией ЭВМ и её составных частей. Учебное пособие состоит из шести глав.

В первой главе даются основные понятия и определения, относящиеся к ЭВМ, приводятся технические и эксплуатационные характеристики, классификации ЭВМ с кратким описанием истории развития, современного состояния и функциональных особенностей различных типов компьютеров (мэйнфреймов, супер ЭВМ, рабочих станций, серверов, персональных компьютеров и т.д.).

Вторая глава посвящена изложению основ функциональной и структурной организации центрального процессора. Подробно рассматриваются архитектуры компьютеров (SISD, SIMD, CISC, RISC, VLIW, EPIC). В главе содержится описание типов данных, используемых в процессорах интеловской архитектуры (IA-32, IA-64), способов адресации данных, структур и форматов команд CISC- и RISC-процессоров, наборы расширения системы команд x86 (MMX, SSE, SSE2, SSE3, SSE4 и др.), обобщенной структуры ЭВМ. Определяется состав и назначение основных устройств процессора; описываются регистровые структуры процессоров IA-32, x86-64, IA-64, особенности современных многоядерных микроархитектур Intel Skylake; AMD Zen, структуры универсальных микропроцессоров Intel, AMD. Особенности организации процессоров семейства «Эльбрус».

В третьей главе описываются принципы организации подсистемы памяти компьютера: рассматривается иерархическая структура памяти компьютера; способы организации кэш-памяти; принципы организации оперативной памяти и методы повышения её пропускной способности. Особое внимание уделяется реализации виртуальной памяти.

В четвёртой главе рассматриваются особенности организации системного интерфейса и ввода/вывода информации: даётся общая характеристика и классификация интерфейсов; описываются способы организации передачи данных и системной организации компьютеров на базе чипсетов Intel и AMD.

В пятой главе рассматриваются устройство, архитектуры и функциональные возможности графических процессоров. Приводится описание архитектур и технологий современных графических процессоров.

В шестой главе кратко описываются архитектуры и классификации многопроцессорных и многомашинных вычислительных систем (MISD, MIMD, SMP, MPP и др.).

В приложении 1 приведены контрольные вопросы и задания для самопроверки.

1 КЛАССИФИКАЦИЯ И АРХИТЕКТУРА ЭВМ. ФУНКЦИОНАЛЬНАЯ И СТРУКТУРНАЯ ОРГАНИЗАЦИЯ

1.1 Определение ЭВМ

Электронная вычислительная машина (далее компьютер) – комплекс технических и программных средств, предназначенных для автоматической обработки информации в процессе решения вычислительных и информационных задач.

Под **системой** понимают любой объект, который одновременно рассматривается и как единое целое, и как объединенная в интересах достижения поставленных целей совокупность разнородных элементов.

Вычислительная система – взаимосвязанная совокупность средств вычислительной техники, включающая не менее двух основных процессоров либо вычислительных машин. Основным процессором называют составную часть ЭВМ, которая выполняет вычисления, предусматриваемые алгоритмами решаемых задач.

Информационная система – взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели. Информационная система немыслима без персонала, взаимодействующего с компьютерами и телекоммуникациями.

Под **архитектурой** ЭВМ понимается общая функциональная и структурная организация машины, определяющая методы кодирования данных, состав, назначение, принципы взаимодействия технических средств и программного обеспечения.

Можно выделить следующие важные для пользователя компоненты архитектуры (рис. 1.1.1):

а) функциональные и логические возможности процессора (система команд, форматы команд и данных, способы адресации, разрядность обрабатываемых слов и т.д.);

б) структурную организацию и принципы управления аппаратными средствами (центральным процессором, памятью, вводом/выводом, системным интерфейсом и т.д.);

в) программное обеспечение (операционная система, трансляторы языков программирования, прикладное программное обеспечение и т.д.).

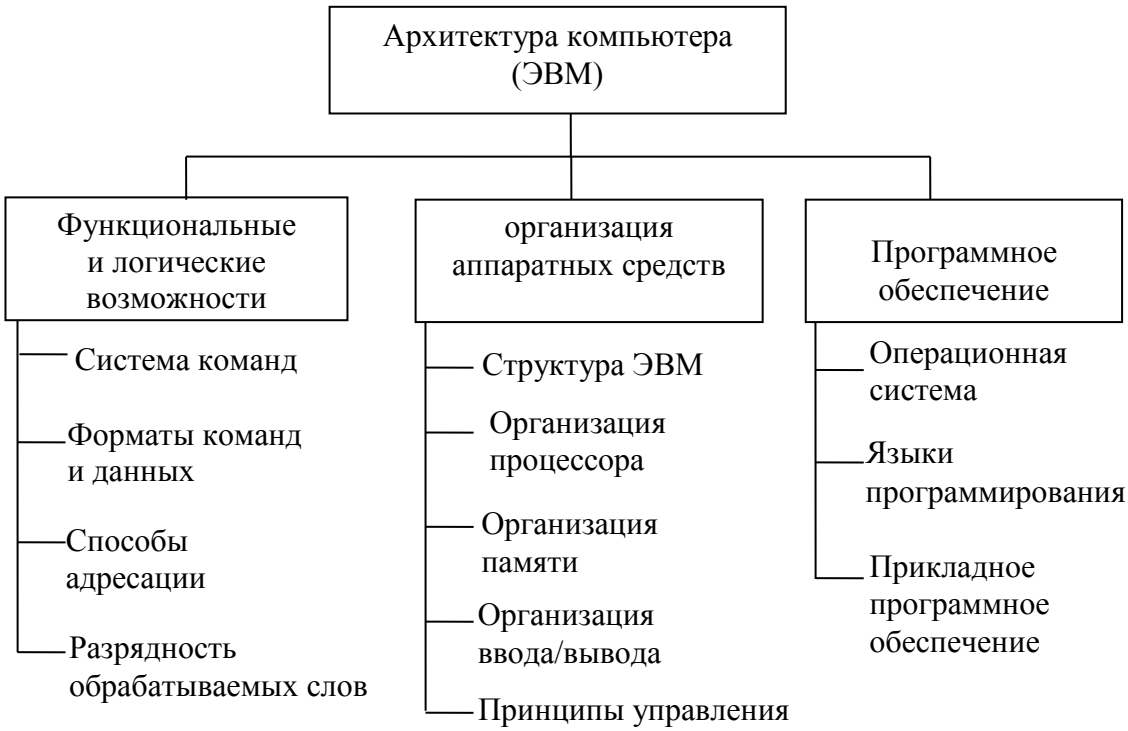


Рис. 1.1.1. Основные компоненты архитектуры ЭВМ

1.2 Функциональная и структурная организация ЭВМ

Существуют два взгляда на построение и функционирование ЭВМ. Первый – взгляд пользователя, не интересующегося технической реализацией ЭВМ и озабоченного только получением некоторого набора функций и услуг, обеспечивающих эффективное решение его задач; второй – разработчика ЭВМ, усилия которого направлены на рациональную техническую реализацию необходимых пользователю функций. С учетом этого обстоятельства и вводятся понятия «функциональная» и «структурная» организация компьютера.

Действительно, с точки зрения пользователя решение любой задачи на ЭВМ требует поэтапного выполнения некоторой последовательности действий: программирования, кодирования, ввода, обработки, документирования. На каждом из этих этапов учет запросов пользователя может потребовать расширения реализуемых ЭВМ функций и услуг, что решается при проектировании ЭВМ и входит в понятие **функциональной организации ЭВМ**.

В результате создается абстрактная модель ЭВМ, описывающая функциональные возможности машины и предоставляемые ею услуги. Функциональная организация ЭВМ в значительной степени определяется предъявляемыми к ней требованиями, уровнем подготовки потенци-

альных пользователей, типом решаемых ими задач, потребностями в развитии компьютера (по емкости памяти, разрядности, составу периферийных устройств и др.).

Предусматриваемые абстрактной моделью функции ЭВМ реализуются на основе реальных физических средств (устройств, блоков, узлов, элементов) в рамках определенной структуры. В общем случае под **структурной организацией ЭВМ** понимается некоторая физическая модель, устанавливающая состав, порядок и принципы взаимодействия основных функциональных частей машины (без излишних деталей их технической реализации).

Функциональная организация ЭВМ играет ведущую роль и в значительной степени определяет структурную организацию машины, хотя и не дает жестких ограничений на конечную техническую реализацию структурных элементов. Вместе с тем функции и структура любого элемента находятся в диалектической взаимосвязи и взаимозависимости. С одной стороны, функциональным назначением устройства (блока, узла) ЭВМ определяется необходимый состав материальных объектов (реальных аппаратных и программных средств) и характер их взаимодействия. С другой стороны, одна и та же функция может быть реализована на совершенно разных технических средствах, а изменение состава или связей между элементами, изменение пропорций между аппаратными и программными средствами может сохранить неизменной функцию системы, сообщив ей новые свойства.

1.2.1 Обобщенная структура ЭВМ и пути её развития

К пяти классическим компонентам компьютера относятся: **устройства ввода, вывода, память, операционный блок, блок управления,** при этом два последних компонента объединяются и называются **процессором**. На рис. 1.2.1 показана **логическая** структура компьютера, которая не зависит от технологии изготовления оборудования: любую часть любого компьютера из прошлого или настоящего можно отнести к одной из этих пяти категорий.

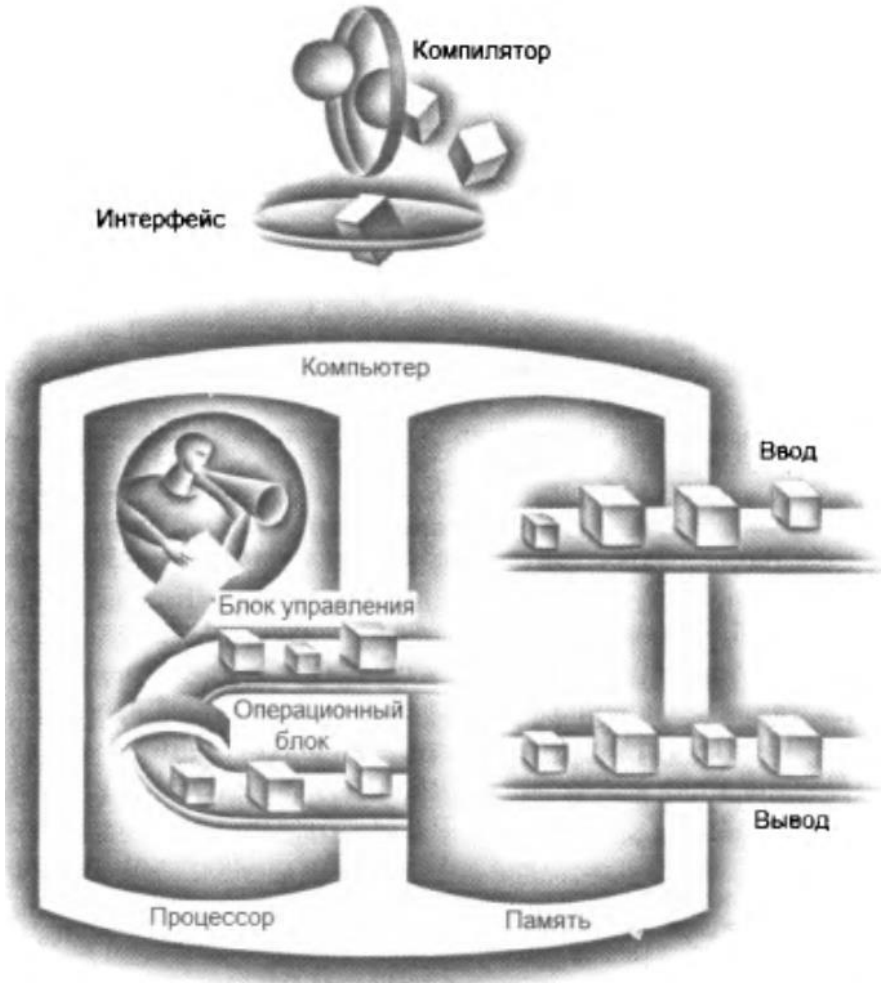


Рис. 1.2.1. Логическая структура компьютера

Процессор получает **инструкции** и **данные** из **памяти**, выполняет над данными арифметические и логические операции (согласно инструкциям), записывает результаты вычислений в память, управляет вычислительными процессами и координирует работу устройств компьютера. Блок **управления** процессора посылает сигналы, определяющие действия операционного блока, памяти, ввода и вывода. Устройство **ввода** записывает данные в память, а устройство **вывода** читает из памяти. Классическим примером устройства **ввода** является клавиатура или мышь, устройством вывода – **монитор**.

Связующим звеном компонентов компьютера является **системная (материнская) плата**. На ней располагаются соответствующие разъемы для подключения памяти, устройств ввода-вывода и сокеты для подключения процессора. Помимо разъёмов и сокетов на системной плате также размещены различные **интегральные микросхемы** или **чипы.** Подробнее устройство материнской платы описано в **главе 4.**

Память является местом хранения выполняемых **программ** и **данных**, необходимых этим программам. Подробнее устройство подсистемы памяти описано в **главе 3.**

Процессор (центральное процессорное устройство, ЦПУ) является то частью компьютера, которая в точности следует инструкциям программ. Процессор осуществляет арифметические операции над числами, подаёт сигналы на активацию устройств ввода-вывода и.т.д. Логически процессор включает **операционный блок** («исполнительная» часть), который выполняет арифметические операции и **блок управления** («мыслительная часть»), который руководит действиями операционного блока, памяти и устройств ввода-вывода в соответствии с предписаниями инструкций программы. Подробнее устройство центрального процессора описано в **главе 2.**

Физическая структура компьютера представлена устройствами, непосредственно осуществляющими работу компьютера.

Устройства ввода-вывода: материнская плата, мышь, клавиатура, монитор, принтер, сканер, дисковод, CD/DVD привод, сетевая карта, звуковая карта.

Память: оперативное запоминающее устройство (ОЗУ), постоянное запоминающее устройство (ПЗУ), кэш-память, жесткий диск.

Блок управления и операционный блок: центральное процессорное устройство.

Также существует отдельное устройство, называемое **видеокарта,** которая включает в себя все логические компоненты компьютера, т.к. содержит **графический процессор**, **память** и устройства ввода-вывода. Подробнее устройство видеокарт и графических процессоров описано в **главе 5.**

Развитие архитектуры неизбежно ведет к развитию структуры ЭВМ. Реализация принципов интеллектуализации, которые все больше определяют развитие архитектуры ЭВМ, возможна при совершенствовании структурной организации, обеспечивающей повышение эффективности вычислительного процесса и, как следствие этого, рост производительности ЭВМ. В конечном счёте условием и критерием развития структуры является рост производительности ЭВМ.

Основной тенденцией в развитии структуры ЭВМ является разделение функций системы и максимальная специализация подсистем для выполнения этих функций.

На рис. 1.2.2 приведена обобщенная структура ЭВМ с выделением путей развития для каждой подсистемы.

Рис. 1.2.2. Обобщенная структура ЭВМ и основные направления ее

1.2.2 Программирование ЭВМ с применением языков низкого и высокого уровней

Обычное приложение, например текстовый процессор или серьезная система управления базами данных, может состоять из нескольких миллионов строк кода и зависеть от использования сложных программных библиотек, реализующих непростые функции, разработанные для поддержки приложений. Для перехода от сложных приложений к простым инструкциям нужны еще несколько уровней программного обеспечения, интерпретирующего или транслирующего высокоуровневые операции в простые компьютерные **инструкции.**

На рис. 1.2.3 показано, что уровни программного обеспечения организованы в виде иерархии, в которой приложение находится в самом внешнем кольце, а **системное программное обеспечение** располагается между аппаратным обеспечением и прикладными программами.

Рис. 1.2.3. Представление иерархических уровней аппаратного и программного обеспечения

Существует множество видов системных программ, но два из них играют главную роль в каждой современной компьютерной системе: это **операционная система** и **компилятор.**

Операционная система является посредником между пользовательской программой и оборудованием и предоставляет различные службы и управляющие функции, такие как:

- обработка основных операций ввода и вывода;
- распределение средств хранения информации и памяти;
- обеспечение защищенного совместного использования компьютера несколькими приложениями;

Компиляторы выполняют другую жизненно важную функцию: трансляцию программы, написанной на языке высокого уровня, например C, C++, Java, в инструкции, которые может выполнять оборудование.

Чтобы осуществлять взаимодействие с электронным оборудованием, нужно передавать ему электрические сигналы. Простейшими сигналами, понятными компьютеру, являются сигналы «включено» (символ 1) и «выключено» (символ 0).

Компьютеры выполняют команды, которые называются **инструкциями**. Инструкции представляют собой наборы битов, которые компьютер «понимает» и принимает на исполнение определенной операции. Например, инструкция 100110010100000 может предписывать компьютеру сложить 2 числа. Подробнее о формате инструкций (команд) описано **в главе 2.**

Первые программисты осуществляли взаимодействие с компьютером посредством двоичных чисел, но это было настолько сложным, что была изобретена новая система записи программы, более близкая к человеку. Используя компьютер, с целью содействия программирования самого компьютера, первопроходцы изобрели программы перевода символьной нотации в двоичную. Первые такие программы были названы **ассемблерами** (сборщиками). Такие программы переводили символьное представление инструкций в их двоичное представление.

Например, если программист писал:

add A,B

то ассемблер мог перевести данную запись в инструкцию:

1000110010100000

которая предписывает компьютеру сложить 2 числа A и B.

В отличие от ассемблера, двоичный язык, понимаемый машиной, называется **машинным** языком.

Несмотря на существенные усовершенствования, язык ассемблера по-прежнему далек от той формы записи, который предпочитают программисты и ученую для решения своих задач. Язык требует от программиста записывать по одной строке для каждой инструкции, которую будет выполнять компьютер, заставляя его мыслить «по-компьютерному». К тому же, при написании программы на ассемблере необходимо использовать команды конкретной архитектуры, т.е. программа, написанная для архитектуры x86, ARM и 8051 на языке ассемблера будет выглядеть по-разному, в то время как программа, написанная **на языке высокого уровня (ЯВУ),** например C или C++ будет выглядеть примерно одинаково для разных архитектур.

ЯВУ и компиляторы для них позволяют вместо ассемблерного кода add A,B написать A+B. На рис.1.3.4 приведена взаимосвязь между этими программами и языками на примере простой программы, меняющий местами соседние элементы массива.

ЯВУ предлагают ряд важных преимуществ. В первую очередь они позволяют программисту размышлять на более естественном для него языке, используя английские слова и алгебраические формы записи, превращающиеся в программы, которые более похожи на текст, чем таблицы криптографических символов (рис. 1.2.4). Более того, они позволяют подстраивать языки дня научных исследований. Кобол – для обработки деловой информации, Лисп – для работы с символами и.т.д.

Вторым преимуществом ЯВУ является повышение производительности труда программистов. В среде разработчиков ПО есть утверждение, что время на разработку программ уменьшается, когда они пишут на языках, которые требуют меньшего количества строк для выражения замысла. Лаконичность – очевидное преимущество ЯВУ над языком ассемблера.

Заключительным преимуществом ЯВУ их независимость от компьютера, на котором они были разработаны, поскольку компиляторы и ассемблеры могут транслировать программы на ЯВУ в язык двоичных инструкций любого компьютера.

Рис. 1.2.4. Программа на С компилируется в программу на языке ассемблера, затем в машинный код

Компиляция и запуск программы

Компилятор превращает программу на языке С в программу на языке *ассемблера*, символьную форму того, что понятно машине. Программы на языке высокого уровня занимают намного меньше строк кода, чем программы на языке ассемблера, поэтому продуктивность работы программиста существенно возрастает. На рис. 1.2.5 показан иерархия трансляции программы на языке С в программный код, понятный

компьютеру. Сначала программа на языке высокого уровня компилируется в программу на языке ассемблера, а затем собирается в объектный модуль на машинном языке. Для разрешения всех ссылок компоновщик объединяет несколько модулей с библиотечными процедурами. Затем загрузчик помещает машинный код в нужное место памяти для выполнения этого кода процессором.

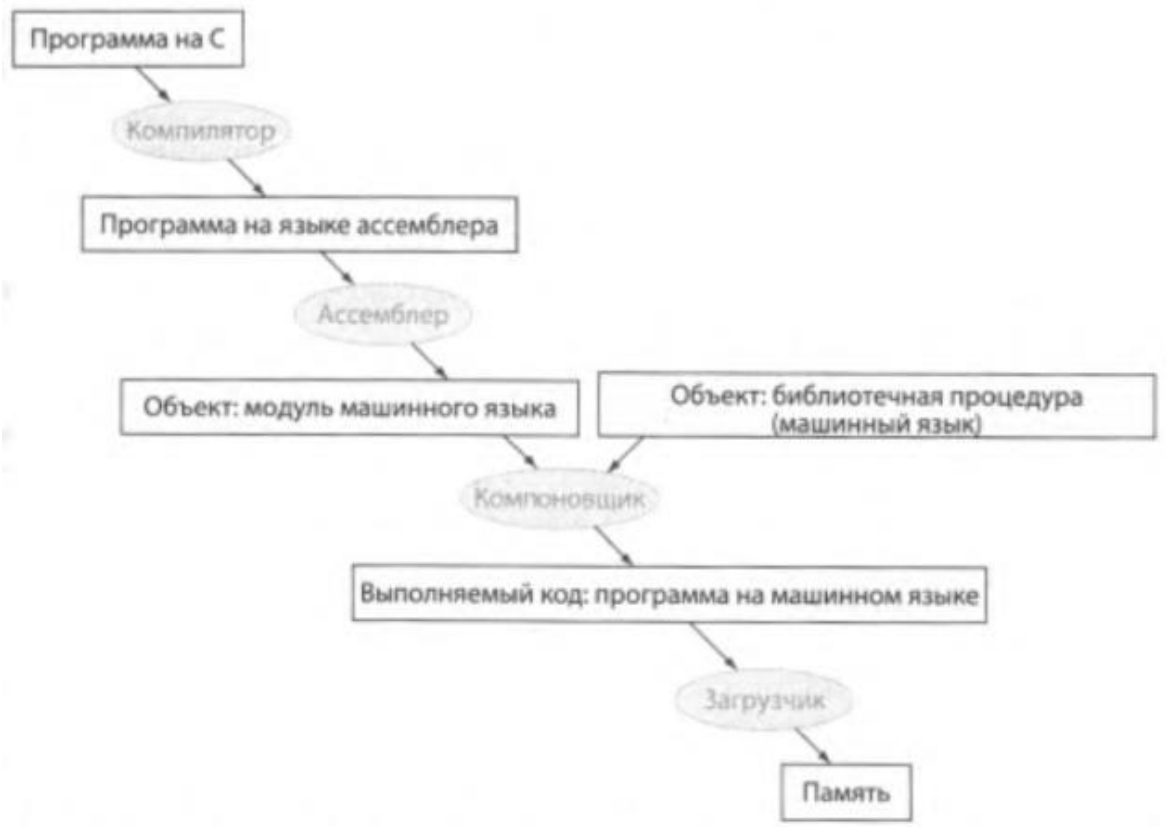


Рис. 1.2.5. Программа на С компилируется в программу на языке ассемблера, затем в машинный код

1.3 Классификация ЭВМ

1.3.1 Общая классификация ЭВМ

В целом, по характеру использования компьютеры делятся на три разных класса: настольные компьютеры, серверы, встраиваемых компьютеры.

Настольные компьютеры составляют наиболее известный вид компьютерной техники. Их типичным представителем является персональный компьютер (ПК). Для ПК характерно предоставление отдельным пользователям хорошей вычислительной производительности при

низкой стоимости и применение компьютерных программ независимых производителей.

Серверы предназначены для выполнения больших объёмов работы, которая может состоять либо из одного комплекса прикладных задач – задачи научного или технического направления, либо из обработки множества мелкий заданий, как, например в случае создания крупного веб-сервера. Доступ к таким ЭВМ осуществляется обычно через сеть. Серверы создаются с применением тех же основных технологий, которые используются и для настольных компьютеров, но с предоставлением существенно больших возможностей по наращиванию вычислительной мощности и средств ввода-вывода. Большое внимание в серверах уделяется функциональной надёжности, по сколько их отказы, по сравнению с отказами обычного ПК, обходятся гораздо дороже.

Серверы имеют широкий спектр цен и возможностей. Самые дешёвые серверы могут стоить чуть больше обычных ПК (за вычетом цены дисплея или клавиатуры). Такие серверы обычно используются для хранения файлов, небольших бизнес-приложений или занимаются обслуживанием сети. На другом конце ценового диапазона находятся **суперкомпьютеры** – класс компьютеров с наибольше производительностью и стоимостью. Компьютеры этого класса предназначены для работы в качестве серверов и в настоящее время содержат до нескольких тысяч процессоров, и, как правило, используют память объёмом в несколько **терабайт (2^{40} или 10^{12} байт)** и внешние хранилища данных объёмом в несколько **петабайт (1000 или 1024 терабайта).** Суперкомпьютеры обычно применяются для выполнения самых сложных научных и технических расчётов, например составления прогнозов погоды, производства нефтеразведочных расчетов, определения структуры белка и решения других крупномасштабных задач.

Встроенные компьютеры представляют собой самый большой класс компьютеров и имеют широкий спектр применения и показателей производительности. К встроенным компьютерам относятся микропроцессоры, которые встречаются в автомобилях, самолетах, сотовых телефонах, смартфонах, телевизорах, приставках, мультиварках и.т.д. Встроенные компьютерные системы сконструированы для запуска одного приложения или набора взаимосвязанных приложений, которые обычно интегрированы с аппаратной частью и поставляются в виде единой системы.

1.3.2 Классификация ЭВМ по назначению

По назначению ЭВМ можно разделить на три группы: **универсальные** (общего назначения), **проблемно-ориентированные** и **специализированные.**

Универсальные ЭВМ предназначены для решения самых различных видов задач: научных, инженерно-технических, экономических, информационных, управленческих и др. В качестве универсальных ЭВМ используются различные типы компьютеров, начиная от суперЭВМ и кончая персональными ЭВМ. Причем одни универсальные ЭВМ могут работать в многопользовательском режиме (в вычислительных центрах коллективного пользования, в локальных компьютерных сетях и т.д.), другие – в однопользовательском режиме.

Проблемно-ориентированные ЭВМ служат для решения более узкого круга задач, связанных, как правило, с управлением технологическими объектами, автоматизированным проектированием, разведкой и добычей нефти, банковским делом, издательской деятельностью и т.д.

Специализированные ЭВМ используются для решения еще более узкого круга задач или реализации строго определенной группы функций. Такая узкая ориентация ЭВМ позволяет четко специализировать их структуру, во многих случаях существенно снизить их сложность и стоимость при сохранении высокой производительности и надежности их работы.

1.3.3 Классификация ЭВМ по функциональным возможностям

По функциональным возможностям и размерам ЭВМ можно разделить на **суперЭВМ, большие** и **микроЭВМ.**

Функциональные возможности ЭВМ обусловливаются основными технико-эксплуатационными характеристиками.

Исторически первыми появились большие ЭВМ, элементная база которых прошла путь от электронных ламп до интегральных схем со сверхвысокой степенью интеграции.

Большие ЭВМ

Большие ЭВМ за рубежом часто называют **мэйнфреймами** (Mainframe). Мэйнфрейм – это высокопроизводительная вычислительная система с большим объемом оперативной и внешней памяти, поддерживающая многопользовательский и многозадачный режимы работы.

Особенности и характеристики современных мэйнфреймов

К ним относятся:

1. **Высокая надежность** (среднее время наработки на отказ оценивается в 12–15 лет) – результат почти 50-летнего совершенствования мэйнфреймов.

2. **Повышенная устойчивость** систем. Мэйнфреймы могут обнаруживать, исправлять и изолировать большинство аппаратных и программных ошибок.

3. **Целостность данных.** В мэйнфреймах используется память с исправлением ошибок.

4. **Рабочая нагрузка мэйнфреймов** может составлять 80–95 % от их пиковой производительности.

5. **Высокая пропускная способность** подсистемы ввода/вывода (канальная архитектура).

6. **Масштабирование** может быть как вертикальным, так и горизонтальным. Вертикальное масштабирование обеспечивается наращиванием до 64 центральных процессоров в одном компьютере. Горизонтальное – реализуется объединением компьютеров в многомашинный (до 32 машин) кластер, выглядящий, с точки зрения пользователя, единым компьютером.

7. **Доступ к данным.** При централизованной обработке информации данные хранятся на одном компьютере, прикладные программы не нуждаются в сборе исходной информации из множества источников (как при распределенной обработке), не требуется дополнительное дисковое пространство для их временного хранения, не возникают сомнения в их актуальности. Все это ведет к снижению стоимости и повышению эффективности обработки.

8. **Защита.** Встроенные аппаратные и программные средства защиты, такие как криптографические устройства, программные продукты защиты операционных систем, обеспечивают совершенную защиту информации.

9. **Непрекращающаяся совместимость –** до сих пор в мэйнфреймах используются приложения, написанные в 70-е гг. Историю полупроводниковых мэйнфреймов принято отсчитывать с появления в 1964 г. универсальной компьютерной системы IBM System/360. За последние десятилетия мэйнфреймам неоднократно предрекали скорую кончину, однако время доказало, что сбить с ног этих «старожилов» не так-то просто.

До сегодняшнего дня бесспорным лидером в производстве мэйнфреймов является корпорация IBM, начиная от серии System/360, затем 370, 390 и до серии z Series. Первые мэйнфреймы этой серии были z800, 890, 900, 990. В 2005 г. IBM объявила о выпуске новых машин z Series

семейства «Z». Очень удачным экземпляром этого семейства была машина z9. В 2008 г. компания IBM выпустила в свет мэйнфрейм System z10 Enterprise Class, представляющий собой 64-процессорную систему, в которой установлены новые процессоры с четырьмя ядрами и частотой 4,4 ГГц. Мэйнфрейм System z10 поддерживает операционные системы z/OS, z/OSe, z/VM, z/VSE, Linux и может обслуживать от сотен до миллионов пользователей в зависимости от приложений.

В 2015 г. компания IBM презентовала новый мэйнфрейм z13, на создание которого ушло 5 лет и около миллиарда долларов США. В процессе работы были использованы инновационные технологии, более 500 новых патентов. Максимальная конфигурация системы z13 включает 141 процессор и 10 ТБ ОЗУ. Процессор содержит 8 ядер, тактовая частота работы – 5 ГГц, технология изготовления – 22 нм.

В июле 2017 г. компания IBM представила систему трансакций z14 с возможностью полного шифрования данных. Разработка системы заняла два года, в ходе которой были опрошены 150 клиентов и директоров по безопасности компании. Система оснащена микропроцессором 5,2 ГГц, способна поддерживать более 12 млрд зашифрованных трансакций в день. Новый мейнфрейм способен зашифровать каждый уровень системы и автоматически отключиться в случае кибератаки. Платформа шифрования в состоянии обеспечить безопасность мировых банковских, медицинских, правительственных инфраструктур и сферы розничной торговли. Отметим, что в настоящее время, 87% платежей по кредитным картам в онлайн-торговле приходится на систему трансакций IBM. По мнению компании, новая система позволит эффективнее бороться с хакерскими атаками, ущерб от которых составит около $8 трлн к 2022 году.

Основными направлениями эффективного применения мэйнфреймов являются пакетная обработка заданий (когда компьютер выполняет работу без участия человека) и обработка заданий в реальном времени (Online), например транзакционные системы, такие как система приобретения железнодорожных билетов, система оплаты по кредитной карте и т.п.

СуперЭВМ

СуперЭВМ – мощные, высокоскоростные вычислительные машины (системы) с производительностью от десятков триллионов (GFLOPS) до нескольких десятков квадриллионов (PFLOPS) операций с плавающей запятой в секунду. Супер ЭВМ выгодно отличаются от больших универсальных ЭВМ по быстродействию числовой обработки, а от специализированных машин, обладающих высоким быстродействием в сугубо

ограниченных областях, – возможностью решения широкого класса задач с числовыми расчетами.

В настоящее время развитие суперкомпьютеров идет по следующим направлениям: векторно-конвейерные компьютеры, параллельные компьютеры с общей памятью, массивно-параллельные системы с распределенной памятью, кластерные системы.

В 2009 г. был преодолен порог производительности суперкомпьютеров в 1 PFLOPS (10^{15} FLOPS). На сегодняшний день в мире насчитывается уже достаточно большое количество суперкомпьютеров, начиная от простых (офисных и персональных) и кончая мощными массивно-параллельными и кластерными системами.

Два раза в год (в июне и ноябре) формируется официальный список пятисот самых мощных суперкомпьютеров мира – Top500. В июне 2018 г. список Top500 возглавила американская система **Summit** (рис. 1.3.1) национальной лаборатории Министерства энергетики США с производительностью (122,3 Пфлопс). Более подробно ознакомиться с рейтингом ТОП-500 можно на сайте https://www.top500.org/.

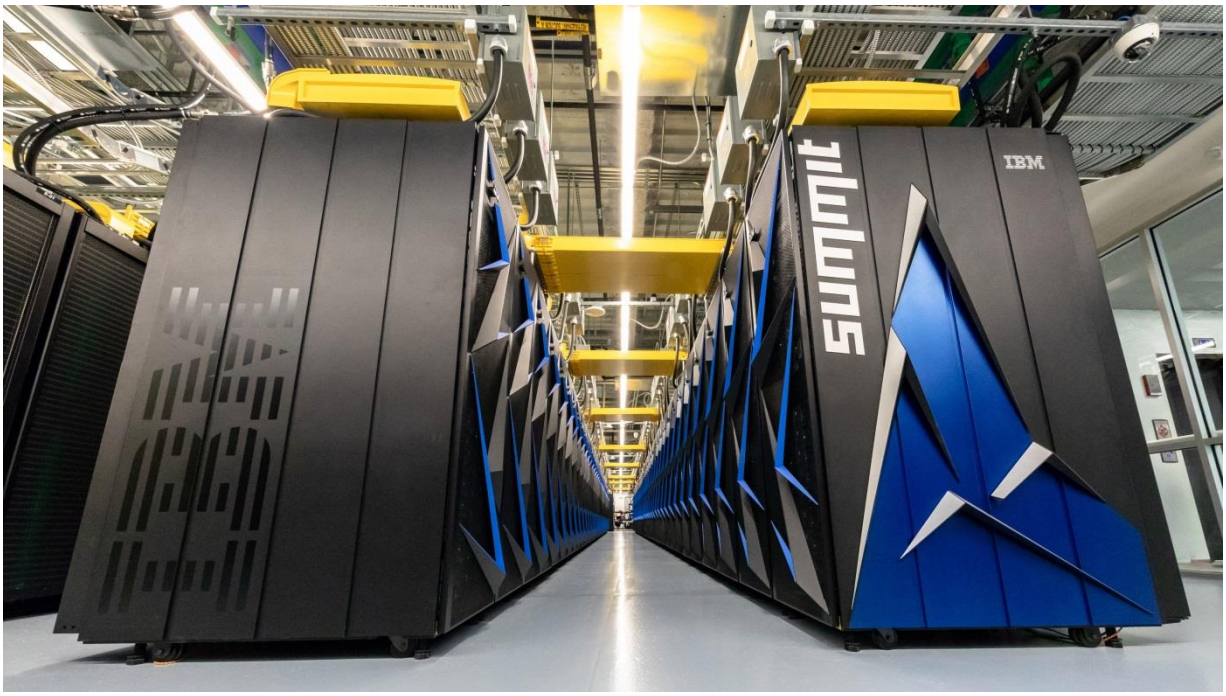


Рис. 1.3.1. Общий вид системы Summit

МикроЭВМ

МикроЭВМ по назначению можно разделить на серверы, рабочие станции, персональные компьютеры, встраиваемые и промышленные микроЭВМ (рис. 1.3.2).

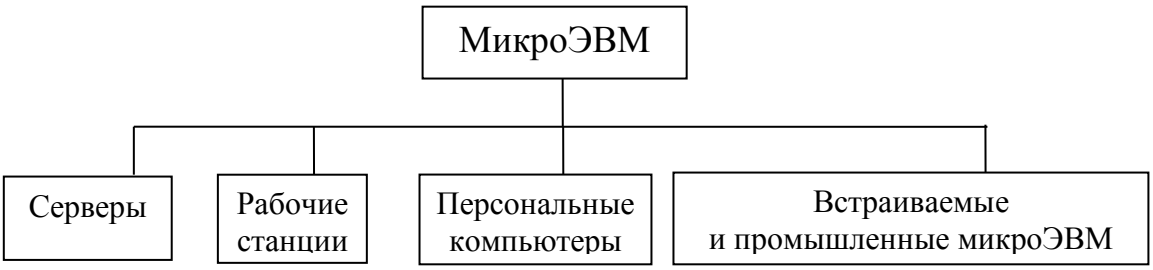


Рис. 1.3.2. Классификация микро-ЭВМ

Серверы

В настоящее время уже редко встретишь офис или предприятие, в котором бы не использовалась компьютерная сеть, время разрозненных персональных компьютеров давно ушло. Однако нагрузка, т.е. уровень сетевого трафика, на различные узлы в сети никогда не бывает равномерно распределенной – на пользовательских компьютерах она всегда меньше, чем на компьютерах, выполняющих служебные функции в сети, **серверах** (от англ. «serve» – служить).

Примером таких функций может быть хранение файлов и обеспечение доступа к ним пользователей (клиентов), маршрутизация потоков данных, управление печатью сетевого принтера, обработка писем электронной почты, рассылка факсов и т.д. Серверами также называются программы, выполняющие эти функции. Ниже под термином «сервер» будет пониматься в первую очередь аппаратное решение.

По функциональному назначению серверы можно подразделить (рис. 1.3.3) на **файл-серверы**, **серверы приложений и баз данных** (чаще всего используются для баз данных и поддержки документооборота), **FTP-серверы** (для удаленного доступа к данным через Internet), **серверы внешних устройств** (печати, сканирования, факсимильной связи) и **Web-серверы**.

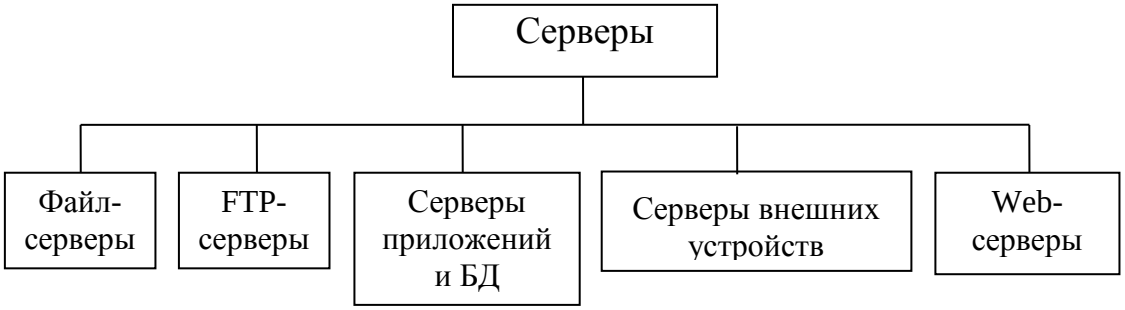


Рис. 1.3.3. Классификация серверов по функциональному назначению

Крупные и мелкие предприятия и офисы обладают вычислительными сетями различной мощности. Кроме того, существуют разные

требования к функциям, выполняемым компьютерной сетью, если одной организации достаточно иметь один файловый сервер, то для другой требуется полный спектр Internet-сервисов, таких как обеспечение получения и отправки электронной почты для всех сотрудников, хостинг (возможность размещения) Web-сайта или FTP-файлового архива. Поэтому не существует «универсального» сервера, способного выполнять любые, совершенно различные задачи одинаково быстро и эффективно.

По функциональным возможностям (мощности) серверы разделяют на **серверы начального**, **среднего** и **корпоративного уровней**. На каждом уровне используются свои способы организации серверов. Для небольшой сети (в рамках рабочих групп – 50 и менее пользователей) функции сервера могут быть возложены на мощный настольный персональный компьютер. Для среднего уровня (50–200 клиентов и малых серверов) могут быть использованы мощные рабочие станции, а для корпоративного (200 и более пользователей) – мэйнфреймы. Кроме того, для каждого уровня иерархии разрабатываются и применяются компьютеры со специальной серверной организацией.

В серверах начального уровня используются как правило от 1 до 8 процессоров, среднего уровня – от 8 до 16 процессоров, корпоративного уровня – от 32 до нескольких сотен процессоров. Количество ядер в процессорах может варьироваться от 2 до 16.

Приведенные классификации весьма условны, потому что в рамках любой серии серверов постоянно появляются модели большей мощности благодаря наращиванию ресурсов и модернизации конфигурации, причем различия внутри одной линейки компьютеров могут быть существенны.

Основными требованиями при проектировании серверов являются:

- **большая мощность** для обеспечения нормальной работы всех запускаемых приложений;
- **масштабируемость**, необходимая при увеличении компьютерной сети предприятия или круга задач, решаемых сервером;
- **отказоустойчивость** для обеспечения надежной работы всех выполняемых программ и сервисов;
- **удобный доступ** к его компонентам с возможностью оперативной или даже «горячей» (автоматической) замены, что очень важно в случае необходимости бесперебойной работы системы.

Сервер начального уровня может собрать хорошо разбирающийся в вычислительной технике человек, но наиболее надежные и сбалансированные системы выпускаются brand-name компаниями, специализирующимися на производстве серверов. Компоненты этих систем обычно

хорошо подобраны друг с другом, что увеличивает эффективность их использования.

Для реализации серверов используются процессоры с архитектурами x86 (CISC), IA-64 (EPIC) и RISC. В качестве основной операционной системы (ОС) может выступать Windows-подобная или Unix-подобная ОС (Unix-серверы).

В последние годы большой популярностью пользуется **операционная система Linux** с открытым исходным кодом, которая была создана в 1991 г. в качестве версии Unix-подобной системы для ПК. Настоящий успех пришел к системе при ее использовании не на настольных ПК, а на серверах. Этому способствует мощная поддержка «китов индустрии» (IBM, HP, Dell, Sun/Oracle), вкладывающих в развитие бесплатной ОС огромные средства.

Облачные вычисления – технология обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как Интернет-сервис. Вся работа с данными происходит на удаленном сервере, а компьютер-клиент лишь интерактивно общается с «облачной фермой», получая от нее картинку. Рядовой пример простого СС – любая веб-почта. В сеть Интернет перебираются приложения, которые раньше представлялись в виде больших программ, продававшихся на DVD: текстовые редакторы, электронные таблицы, графические пакеты, игры и т.д.

По мере того, как компании станут все шире использовать такие модели вычислений, как «облака», будет расти спрос на **микросерверы** – малогабаритные серверы с малым потреблением энергии и невысокой ценой.

Новый класс устройств – **миниатюрные домашние серверы** – становится обыденным явлением не только дома, но и в офисах малых и мобильных компаний. В настоящий момент на рынке можно найти около полудюжины производителей, предлагающих портативные и не очень дорогие домашние серверы для мобильных пользователей. Основное их отличие – возможность быстрого перемещения вслед за «летучей» рабочей группой или домашним офисом. Поскольку все взаимодействия с ними обеспечиваются по беспроводной связи, то их применение экономит пользователям много времени и средств, которые бы они раньше потратили на прокладку линий связи. Домашний сервер берет на себя организацию передачи различных типов информации внутри небольшой локальной компьютерной сети.

По конструктивному исполнению серверы могут быть **башенными, стоечными** и **модульными.**

Модульные серверы

К категории модульных серверов по классификации аналитической компании Gartner относятся блейд-серверы (лезвия) и многоузловые сверхплотные серверные системы. Оба типа модульных серверов используют для размещения стоечные шасси, где помимо самих серверов устанавливаются устройства хранения данных и сетевое оборудование. При этом их можно легко извлекать из шасси и добавлять в него.

Организация блейд-серверов основывается на концепции адаптивной инфраструктуры, которая предусматривает гибкость, экономичность и оперативность подстройки под быстро меняющиеся требования пользователей.

По определению аналитической компании IDC **блейд-сервер** – это модульная одноплатная компьютерная система, включающая процессор и память (рис. 1.3.4). Лезвия вставляются в специальное шасси (или полку) с объединительной панелью (back plane), обеспечивающей им подключение к сети и подачу электропитания. Это шасси с лезвиями, по мнению IDC, является блэйд-системой. Оно выполнено в конструктиве для установки в стандартную 19-дюймовую стойку и, в зависимости от модели и производителя, занимает в ней 3U, 6U или 10 U (один U – unit, монтажная единица, равная 1,75 дюйма, 1 дюйм равен 2,54 см). За счет общего использования таких компонентов, как источники питания, сетевые карты, жесткие диски и блоки охлаждения, блэйд-серверы обеспечивают более высокую плотность размещения вычислительной мощности в стойке по сравнению с обычными тонкими серверами высотой 1U и 2U. Блейды большинства изготовителей монтируются в шасси вертикально.

К преимуществам использования блейд-серверов можно отнести следующие:

- уменьшение занимаемого объема;
- уменьшение энергопотребления и выделяемого тепла;
- уменьшение стоимости и повышение надежности системы питания и охлаждения;
- повышение удобства управления системой;
- высокая масштабируемость;
- высокая гибкость;
- сокращение количества коммутационных проводов.

Рисунок 1.3.4. Общая конструкция блейд-сервера

Рабочие станции

Рабочая станция (Work station), по определению экспертов IDC (International Data Corporation), – это однопользовательская система с мощным одним или несколькими процессорами и многозадачной ОС, имеющая развитую графику с высоким разрешением, большую дисковую и оперативную память и встроенные сетевые средства.

Изначально рабочие станции (WS) ориентируются на **профессиональных пользователей**. Этот вид ЭВМ появился на компьютерном рынке почти одновременно с персональными компьютерами (ПК) и в целом опережает их по своим вычислительным возможностям. В отличие от ПК, ориентированных на самый широкий круг пользователей, рабочие станции предназначены для корпоративного сектора рынка. Ориентация на корпоративное использование и на профессионального пользователя позволяет во многих случаях применять более совершенные и дорогостоящие аппаратные средства.

Рабочие станции, используя те же процессоры и практически не отличаясь от ПК по внешнему виду, **обладают рядом специфических характеристик**, не свойственных ПК, таких, как поддержка профессиональной двух- и трехмерной графики и многодисковых конфигураций, большой объем и быстродействие жесткого диска, использование двух процессоров (в старших моделях), применение памяти с коррекцией

ошибок. Благодаря этому у них выше производительность, надежность и больше графических возможностей, чем у ПК.

Современная рабочая станция – это не просто большая вычислительная мощность. Это тщательно сбалансированные возможности всех подсистем машины, чтобы ни одна из них не стала «бутылочным горлышком», сводя на нет преимущества других. Кроме того, каждая WS, как правило, предназначена для решения определенного класса задач, поэтому в ней используется наиболее эффективное для этого класса аппаратное и программное оснащение.

Традиционными областями применений рабочих станций является работа с компьютерной графикой (трехмерная анимация, создание трехмерных моделей, визуализация различных процессов), автоматизированное проектирование, издательская деятельность. Также WS применяются для осуществления сложных расчетов в самых различных областях науки, при моделировании различных процессов. В этом качестве WS вытеснили с рынка дорогостоящие миниЭВМ, которые как класс компьютеров прекратили свое существование.

Персональные компьютеры

Персональные компьютеры (ПК) – это однопользовательские микро ЭВМ, удовлетворяющие требованиям общедоступности и универсальности применения.

Для удовлетворения этим требованиям персональный компьютер **должен иметь следующие характеристики:**

- невысокую стоимость, находящуюся в пределах доступности для индивидуального покупателя;
- простоту использования;
- возможность индивидуального взаимодействия пользователя с компьютером без посредников и ограничений;
- высокие возможности по переработке, хранению и выдаче информации;
- гибкость архитектуры, обеспечивающую ее адаптивность к разнообразным применениям в сфере управления, науки, образования, в быту;
- высокую надежность, простоту ремонта и эксплуатации;
- «дружественность» операционной системы;
- наличие программного обеспечения, охватывающего практически все сферы человеческой деятельности.

На сегодняшний день **большинство настольных ПК базируется на x86 процессорах** с основной операционной системой из семейства Windows. Эта платформа процессоров является де-факто самой распространенной, демократичной, во многих своих ипостасях гораздо более

дешевой и универсальной. Данное направление имеет большое количество клонов, т.е. аналогичных компьютеров, выпускаемых различными фирмами США, Западной Европы, России, Японии и др.

Другая платформа представлена довольно популярными на Западе компьютерами **Macintosh фирмы Apple**. Они занимают на мировом рынке компьютеров довольно узкую нишу, тем не менее удерживают ее за собой в течение уже очень большого промежутка времени с переменным успехом, но все же достаточно стабильно.

Программное обеспечение, операционная система, даже «идеология использования компьютера» – все это в указанных платформах очень сильно разнится. На сегодняшний день формальным отличием является только операционная система (MacOSX, имеющая Unix-подобное ядро). С недавнего времени в компьютерах Macintosh используются процессоры x86 вместо RISC-процессоров Power PC.

По способу использования ПК можно разделить на два основных класса: **стационарные** и **переносные** ПК (рис. 1.3.5).

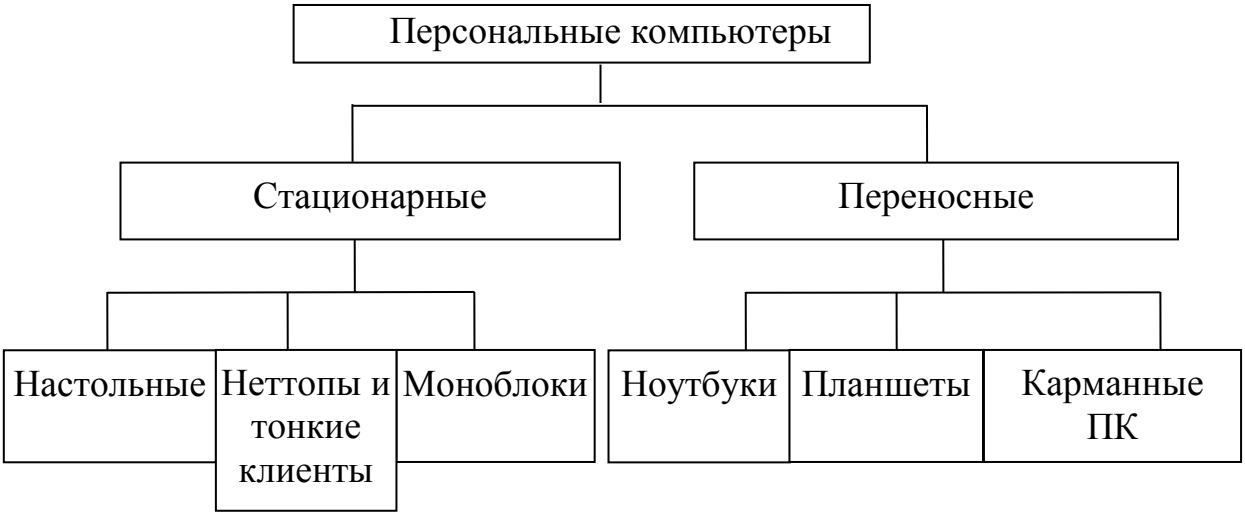


Рис. 1.3.5. Классификация персональных компьютеров по способу использования

Стационарные ПК

Неттопы – это ориентированные на работу в Интернете настольные ПК на базе процессоров Intel Atom; представляют собой простые в использовании и компактные устройства, имеющие оптимальную производительность для использования всех технологий Интернета. Они отличаются надежностью и гибкими возможностями беспроводной связи. Эти устройства предназначены для обучения, просмотра видео и

фотографий, общения в социальных сетях, Интернет-телефонии, работы с электронной почтой, обмена сообщениями, просмотра сайтов и решения других задач.

Мировой рынок настольных ПК в последние годы переживает острейший кризис из-за спада спроса на свою продукцию. Классические десктопы в домашних условиях активно вытесняются, с одной стороны, неттопами, а с другой – стремительно совершенствующимися игровыми приставками; в офисах все чаще устанавливаются «тонкие клиенты».

Тонкий клиент (thin client) в компьютерных технологиях – это бездисковый компьютер-клиент в сетях с клиент-серверной или терминальной архитектурой, который переносит все или большую часть задач по обработке информации на сервер. Тонкий клиент в большинстве случаев обладает минимальной аппаратной конфигурацией.

Моноблок – это компьютер, собранный в одном корпусе с монитором. В настоящее время, когда используются большие плоские ЖК-панели, моноблок внешне очень похож на монитор и имеет больше органов управления. В принципе, близкий аналог моноблока — монитор, к задней панели которого прикреплен неттоп.

Основное преимущество моноблока состоит в том, что по размерам, компоновке и весу он не сильно отличается от обычного ЖК-монитора, но при этом представляет собой вполне полноценный компьютер. Приобретая моноблок, пользователь в первую очередь сильно экономит в занимаемом системой месте. У моноблока: полностью отсутствует системный блок, для него не нужно отводить пространство под/на/в столе. Моноблок легко двигать и относительно легко переносить внутри помещения. Полностью отсутствуют соединительные провода, большинство домашних моделей имеют беспроводные клавиатуру и мышь. Большинство современных моноблоков используют безвентиляторные блоки питания, они меньше компьютерных по габаритам и совсем не шумят.

Минусы моноблока в сравнении с полноценным настольным компьютером во многом совпадают с минусами ноутбуков. Отведенное для системных компонентов место очень ограничено (по глубине корпуса его вообще практически нет), поэтому возникают некоторые проблемы даже с размещением компонентов, а уж сделать мощную систему охлаждения и вовсе невозможно.

Переносные ПК

Увеличивающийся спрос на мобильные ПК объясняется ростом их производительности и снижением цены. Производители компьютерной

техники быстро переориентировались на производство мобильных ПК. В 2009 г. мировой объем поставок ноутбуков превысил объем поставок настольных ПК.

Требования к переносным компьютерам сильно отличаются от требований к настольным ПК. **Мобильные ПК** должны иметь:

- миниатюрные внутренние компоненты и периферийные устройства;
- автономное электропитание;
- низкое энергопотребление;
- малые габариты и вес.

Переносные ПК по своим конструктивным особенностям можно разделить: ноутбуки, планшеты и карманные ПК (сотовые телефоны, смартфоны). Существует термин **лэптоп** (laptop – наколенный), который применяется как к ноутбукам, так и к планшетным ПК. К ноутбукам обычно относят лэптопы, выполненные в раскладном форм-факторе.

Стандарты защиты устройств

Для всех категорий переносных устройств существует стандарт защиты **IP (Ingress Protection Rating)** - классификатор степеней защиты электронных устройств от попадания внутрь воды и твердых предметов различного размера. В зависимости от размера посторонних предметов устанавливается степень защиты, которой соответствует то или иное устройство. Оборудование, прошедшее этап стандартизации маркируется международным знаком защиты «IP» затем присваивается код из двух цифр, первая из них характеризует устройство по степени защиты от попадания внутрь твердых предметов, а вторая от попадания воды или влаги. В итоге получается код вида IPXX, где XX – это цифра соответствия по шкале норм, либо просто символ X, если степень соответствия не установлена.

Стандарты соответствия по первой цифре.

Характеризует устройство по степени соприкосновения частей тела (рук, пальцев) или предметов, находящихся в руках человека с внутренними частями, находящимися под защитной оболочкой или корпусом, а также от попадания внутрь посторонних предметов, мусора или пыли. Маркируется от нуля до шести.

Если присвоена цифра 0 – означает, что корпус не обеспечивает защиту от попадания внутрь посторонних предметов, пыли или мусора, а также не защищает от опасных частей устройства.

Цифра 1 говорит о том, что защитная оболочка защищает пользователя от опасных частей механизма при касаниях тыльной стороной руки, 2 — пальцем, 3 — инструментом, 4, 5 и 6 — проволокой.

Первая цифра 1,2,3,4, говорит о том, что оболочка обеспечивает защиту от попадания внутрь твердых предметов с диаметром соответственно 50, 12,5, 2,5 и 1,0 мм.

Цифра 5 говорит о частичной защите от попадания внутрь пыли, а 6 о полной защите от попадания пыли.

Вторая цифра в маркировке.

Цифра 0 говорит о том, что устройство не имеет защиты от попадания внутрь воды и влаги.

Цифра 1 говорит о том, что корпус устройства защищает от вертикально капающих капель воды.

Цифра 2 говорит о защите от попадания внутрь вертикаль капающих капель воды, при наклоне устройство до 15 градусов.

Цифра 3 свидетельствует о защите устройства от капель в виде дождя.

Цифра 4 свидетельствует о защите устройства от массивных брызг воды.

Цифра 5 свидетельствует о защите устройства от струй воды.

Цифра 6 свидетельствует о защите устройства от массивных струй воды.

Цифра 7 свидетельствует о защите устройства от попадания внутрь воды и влаги при частичном или непродолжительном погружении в воду.

Цифра 8 свидетельствует о полной защите устройства при длительном погружении в воду.

Ноутбуки

Все **ноутбуки** (notebook) классифицируются на несколько типовых разновидностей по размеру диагонали дисплея, назначению, компоновке составных узлов, функциональным возможностям, габаритам, весу и другим отличиям. К основным типам ноутбуков можно отнести: **«замену настольного ПК»** (Desktop Replacement), **массовые ноутбуки, ультрабуки, нетбуки.**

В качестве **замены настольного ПК** обычно позиционируются ноутбуки с диагональю экрана 17 дюймов и выше. Габариты и вес (от 3 кг и выше) портативных компьютеров весьма значительны, что делает их неудобными в переноске. Однако относительно большой размер дисплея обеспечивает более комфортную работу, а объемистый корпус позволяет установить мощные компоненты и обеспечить им достаточное охлаждение. Такие ноутбуки имеют встроенные жесткий диск, аккумулятор, CD или DVD-привод, порты ввода/вывода. Снаружи подсоединяется блок питания, как у всех других ноутбуков.

Массовые ноутбуки (специального названия для данной категории ноутбуков не предусмотрено) имеют диагональ экрана 14'-16', их вес обычно укладывается в 2–3 кг, толщина оказывается чуть меньше ноутбуков «замена настольного ПК». Обычно эти модели оснащены встроенными жестким диском и оптическим накопителем.

Ультрабуки (ultrabooks) – тонкий и легкий ноутбук, обладающий ещё меньшими габаритами и весом по сравнению с обычными ноутбуками, но при этом – большей частью характерных черт полноценного ноутбука. Термин стал широко распространяться в 2011 году, после того как корпорация Intel презентовала новый класс мобильных ПК – ультрабуки.

Первоначально концепция мобильного компьютера, более компактного и лёгкого, чем обычный ноутбук, появилась в 1996 году, когда корпорация Toshiba выпустила семейство ноутбуков Toshiba Libretto. Этот класс компьютеров получил наименование субноутбуки. С тех пор в течение 15 лет субноутбуки постоянно развивались в направлении снижения габаритов и цены и увеличения вычислительной мощности и длительности автономной работы от встроенной аккумуляторной батареи.

15 января 2008 года Стив Джобс провёл презентацию нового сверхлёгкого субноутбука Apple MacBook Air, выполненного в сверхтонком алюминиевом корпусе и не имевшего аналогов на тот момент. После начала продаж выяснилось, что данный субноутбук имеет повышенный спрос у потребителей, и вскоре стали появляться аналоги от других производителей ноутбуков: Dell Adamo, Lenovo ThinkPad X300, Samsung 900X3A, Sony Vaio Y.

В мае 2011 года появился новый класс мобильных ПК – ультрабуки, который является дальнейшим эволюционным развитием классических субноутбуков и во многом использует идеи, реализованные в сверхтонком ноутбуке от Apple, MacBook Air.

Нетбуки (netbooks) как отдельная категория ноутбуков были выделены из категории субноутбуков в 2008 г. компанией Intel. Размер диагонали экрана нетбуков – от 7' до 12,1'. Нетбуки ориентировались на просмотр веб-страниц, работу с электронной почтой и офисными программами. Для этих ноутбуков были разработаны специальные энергоэффективные процессоры Intel Atom, VIA C7, VIA nano, AMD Geode. Малый размер экрана, небольшая клавиатура и низкая производительность подобных устройств компенсировались умеренной ценой и относительно большим временем автономной работы. Габариты обычно не позволяли устанавливать в нетбук дисковод оптических дисков, однако

Wi-Fi-адаптер являлся обязательным компонентом. Столкнувшись с конкуренцией со стороны ультрабуков и планшетных ПК, натиск последних выдержали лишь компнаии Asustek и Acer, которые продавали свои нетбуки плоть до конца 2012 года в основном на разививающихся рынках Южной Азии и Южной Африки. Эра нетбуков закончилась в 2012 г. В 2013 г. распродавались только их запасы.

Ноутбуки-трансформеры. В 2015 году компания Microsot неожиданно для многих, кроме планшета Surface Pro 4, представила также ультрабук Surface Book (рис. 1.3.6). С технической точки зрения аппарат похож на планшет Microsoft. Тут используется корпус из того же магниевого сплава, а дисплей также располагает специальным слоем для работы со стилусом. К слову, перо Surface Pen поставляется в комплекте с новинкой. Необычным выглядит конструкция петель. Несмотря на отключаемую планшетную часть, инженеры Microsoft наделили устройство возможностью раскрыть дисплей на 360°.

Однако Microsoft называет новинку просто ноутбуком. В этом случае в первую очередь обращает на себя внимание дисплей диагональю 13,5 дюйма. У него крайне необычное для ноутбуков соотношение сторон (3:2) и разрешение (3000 x 2000 точек).

Рис. 1.3.6. Ноутбук-трансформер Surface Book от Microsoft

Планшеты

Планшетные ПК (Tablet PC) – устройства с жидкокристаллическими сенсорными дисплеями, позволяющими работать без использования клавиатуры. Корпорация Microsoft предложила новый тип мобильных компьютеров еще в 2002 г. Отдельные технологические решения, входящие в состав платформы, давно известны. Однако корпорация Microsoft, объединив достижения ноутбукостроения, оцифровки графики, беспроводной связи и усовершенствовав программы для распознавания рукописного текста и голоса, сумела создать новую, логически завершенную концепцию работы с информацией в «полевых» условиях.

В планшетный компьютер можно ввести данные, «написав» их специальным пером прямо на поверхности монитора, причем не стараясь выводить печатные символы, а просто так, как вы делали бы это в бумажном блокноте. Более того, информацию можно просто надиктовать в микрофон планшетного компьютера, а соответствующая программа переведет речь в обычный текст. Беспроводный доступ к локальным компьютерным сетям поможет тут же передать информацию по назначению или запросить нужные данные, в том числе из сети Интернет. Конструктивно планшетный компьютер – это дисплей, под которым спрятана элементная база обычного современного ноутбука: процессор, жесткий диск, оперативная память и модули беспроводного доступа. Некоторые модели снабжены собственной клавиатурой. Планшетные компьютеры более легкие и мобильные, чем обычные ноутбуки. Они обладают отличными демонстрационными возможностями.

Электронные книги относятся к новому поколению универсальных устройств для чтения текстовых документов. Устройство снабжено «бумагоподобным» экраном от компании E-Ink, что гарантирует высочайшее качество отображения текстовых документов. Основным отличием данной группы компьютерных устройств от планшетных ПК является ограниченная функциональность, а также существенно большее время автономной работы. Последнее достигается за счет использования технологии «электронной бумаги». Дисплей, выполненный по этой технологии, отображает лишь несколько оттенков серого цвета, но при этом отражает свет (сам не светится) и потребляет энергию только для формирования изображения (перелистывания страницы).

Карманные ПК

Карманные устройства с диагональю экрана менее 7' выделяют в специальную категорию «наладонных компьютеров» (hand held PC),

которые можно подразделить на **карманные ПК (КПК), смартфоны** и **коммуникаторы**.

Карманный персональный компьютер (КПК) – портативное вычислительное устройство, обладавшее широкими функциональными возможностями, начиная от чтения электронных книг и кончая выполнением офисных приложений. Изначально КПК предназначались для использования в качестве электронных органайзеров. На английском языке словосочетание «карманный ПК» (**Pocket PC**) являлось **торговой маркой** фирмы Microsoft, т.е. относилось лишь к одной из разновидностей КПК, а не обозначало весь класс устройств. Словосочетание **Palm PC** («наладонный компьютер») также являлось конкретной торговой маркой. Для обозначения всего класса устройств в английском языке использовалась аббревиатура **PDA – Personal Digital Assistant** – «личный цифровой секретарь». К карманному ПК, оснащенному хост-контроллером USB, можно было напрямую подключать различные USB-устройства, в том числе клавиатуру, мышь, жесткий диск и флэш-накопитель. Основными операционными системами для КПК являлись: Windows Mobile (ранее Pocket PC и Windows CE) фирмы Microsoft; Palm OS фирмы Palm Source. Начиная с 2006 г., объемы поставок КПК стали постоянно снижаться и в 2008 г. КПК были практически вытеснены **смартфонами** и **коммуникаторами**.

Смартфоны и коммуникаторы. В настоящее время не существует четкого разграничения между смартфонами и коммуникаторами, поскольку функциональность обоих классов устройств примерно одинакова. Различные эксперты и производители по-разному трактуют эти термины. Часто применяется так называемый «исторический подход», который заключается в следующем: если устройство ведет свою родословную от КПК – это коммуникатор, а если от мобильных телефонов – это смартфон. Таким образом, **смартфон** (smartphone – интеллектуальный телефон) – мобильный телефон с расширенной функциональностью, сравнимой с КПК; **коммуникатор** (communicator, PDA Phone) – карманный персональный компьютер, дополненный функциональностью мобильного телефона. В рамках этого подхода изначально под коммуникаторами подразумевались устройства с сенсорным экраном (мог быть дополнен клавиатурой). Устройства, использовавшие для ввода информации исключительно полноразмерную (QWERTY) клавиатуру и/или цифровую клавиатуру (аналог телефонной), назывались смартфонами. В большинстве случаев позиционирование устройства зависело от производителя (обычно устройства с сенсорным экраном относились к коммуникаторам, а к смартфонам относились устройства без

такого экрана). Часть специалистов разделяло коммуникаторы и смартфоны соответственно наличием или отсутствием полноразмерной (QWERTY) клавиатуры (виртуальной или физической). В настоящее время в смартфонах и коммуникаторах используется небольшое количество (ОС, Android, Windows Phone).

В 2012 г. на рынке появилась новая разновидность смартфонов под названием фаблеты. **Фаблет (phablet)** – это сматрфон, имеющий диагональ экрана от 5,5 дюймов, большие габариты и более долгий срок использования.

1.4 Технические и эксплуатационные характеристики ЭВМ

Производительность компьютера

Основным техническим параметром ЭВМ является её **производительность**. Этот показатель определяется архитектурой процессора, иерархией внутренней и внешней памяти, пропускной способностью системного интерфейса, системой прерывания, набором периферийных устройств в конкретной конфигурации, совершенством ОС и т.д.

Различают следующие виды производительности:

- **пиковая** (предельная) – это производительность процессора без учета времени обращения к оперативной памяти (ОП) за операндами;
- **номинальная** – производительность процессора с ОП;
- **системная** – производительность базовых технических и программных средств, входящих в комплект поставки ЭВМ;
- **эксплуатационная** – производительность на реальной рабочей нагрузке, формируемой в основном используемыми пакетами прикладных программ общего назначения.

Методы определения производительности разделяются на три основных группы:

- **расчетные**, основанные на информации, получаемой теоретическим или эмпирическим путем;
- **экспериментальные**, основанные на информации, получаемой с использованием аппаратно-программных измерительных средств;
- **имитационные**, основанные на моделировании и применяемые для сложных ЭВМ.

Основные единицы оценки производительности:

- **абсолютная**, определяемая количеством элементарных работ, выполняемых в единицу времени;
- **относительная**, определяемая для оцениваемой ЭВМ относительно базовой в виде индекса производительности.

Для каждого вида производительности применяются следующие традиционные методы их определения.

Пиковая производительность (быстродействие) определяется средним числом команд типа «регистр–регистр», выполняемых в одну секунду, без учета их статистического веса в выбранном классе задач.

Номинальная производительность (быстродействие) определяется средним числом команд, выполняемых подсистемой «процессор–память» с учетом их статистического веса в выбранном классе задач. Она рассчитывается, как правило, по формулам и специальным методикам, предложенным для процессоров определенных архитектур, и измеряется с помощью разработанных для них измерительных программ, реализующих соответствующую эталонную нагрузку.

Для данных типов производительностей используются следующие единицы измерения:

- MIPS (Mega Instruction Per Second) – миллион команд в секунду;
- MFLOPS (Mega Floating Operations Per Second) – миллион операций над числами с плавающей запятой в секунду;
- GFLOPS (Giga Floating Operations Per Second) – миллиард операций над числами с плавающей запятой в секунду;
- TFLOPS (Tera Floating Operations Per Second) – триллион операций над числами с плавающей запятой в секунду;
- PFLOPS (Peta Floating Operations Per Second) – квадриллион операций над числами с плавающей запятой в секунду.

Системная производительность измеряется с помощью синтезированных типовых (тестовых) оценочных программ, реализованных на унифицированных языках высокого уровня. Унифицированные тестовые программы используют типичные алгоритмические действия, характерные для реальных применений, и штатные компиляторы ЭВМ. Они рассчитаны на использование базовых технических средств и позволяют измерять производительность для расширенных конфигураций технических средств. Результаты оценки системной производительности ЭВМ конкретной архитектуры приводятся относительно базового образца, в качестве которого используются ЭВМ, являющиеся промышленными стандартами систем ЭВМ различной архитектуры. Результаты оформляются в виде сравнительных таблиц, двумерных графиков и трехмерных изображений.

Эксплуатационная производительность оценивается на основании использования данных о реальной рабочей нагрузке и функционировании ЭВМ при выполнении типовых производственных нагрузок в основных областях применения. Расчеты делаются главным образом

на уровне типовых пакетов прикладных программ текстообработки, систем управления базами данных, пакетов автоматизации проектирования, графических пакетов и т.д.

Была создана целая **процедура тестирования True Performance Initiative** (процедура измерения реальной производительности – TPI). Методика TPI состоит в измерении эксплуатационной производительности в трех разделах: Productivity – программные приложения; Visual Computing – компьютерная визуализация; Gaming – компьютерные игры.

Для первого раздела используются тесты: Sysmark2007, Mathematica 6, 3ds Max 9 (SPECapc) и др.; для второго – Photoshop CS 3, After Effects CS3, Win RAR 3.7; для третьего – 3DMark2006, Quake 4 и др.

Энергоэффективность процессора

В последнее время при сравнении процессоров пользуются отношением производительности к энергопотреблению, которое получило название **энергоэффективность процессора**. Разработчики процессоров предложили оценивать производительность (P) как произведение тактовой (рабочей) частоты процессора (f) на величину k, определяющую количество инструкций, исполняемых процессором за один такт:

$$P = f \cdot k.$$

Получается, что увеличить производительность можно, поднимая частоту и/или увеличивая количество инструкций, выполняемых за один такт. Первый подход ведет к увеличению энергопотребления, а второй требует использования определенной микроархитектуры процессора (многоядерной), в которой заложены различные технологии, направленные на повышение количества инструкций, выполняемых процессором за один такт.

Что касается энергопотребления (W), то оно вычисляется как произведение тактовой частоты (f) процессора на квадрат напряжения U, при котором функционирует процессорное ядро, и некоторую величину Cd (динамическая емкость), определяемую микроархитектурой процессора и зависящую от количества транзисторов в кристалле и их активности во время работы процессора:

$$W = f \cdot U^2 \cdot Cd.$$

Из приведенных формул вытекает следующее соотношение, определяющее энергоэффективность процессора:

$$P/W = k / (U^2 \cdot Cd).$$

Из формулы следует, что для получения наилучшего показателя разработчикам необходимо работать над оптимизацией микроархитектуры с целью улучшения функциональности исполнительных блоков, при этом не допуская чрезмерного увеличения динамической емкости. Что касается тактовой частоты, то, как показывают приведенные выкладки, на рассматриваемое соотношение она вообще не влияет. Напряжение питания ядра зависит не столько от микроархитектуры, сколько от технологических особенностей изготовления процессора.

Любой современный кристалл процессора состоит из огромного количества транзисторов, исчисляемого миллионами, необходимого для достижения высокой производительности процессора. Уменьшение размеров транзистора ведет к уменьшению напряжения питания, что, в свою очередь, снижает энергопотребление, к увеличению скорости работы и плотности размещения транзисторов на кристалле. Поэтому со времени создания первой интегральной микросхемы в 1959 г. развитие микроэлектроники идет по направлению уменьшения размеров транзисторов и одновременного увеличения плотности их размещения на кристалле. Для оценки этих параметров была введена специальная характеристика «**Норма технологического процесса производства полупроводниковых кристаллов**», измеряемая в нанометрах (нм). В недалеком прошлом (конец 90-х гг.) кристаллы процессоров изготавливались по 130 нм нормам, затем по 90 нм, 65 нм. С 2008 г. используются 45 нм, с 2010 г. – 32 нм, с 2012 г. – 22 нм, а с 2014 г. – 14 нм нормы технологического процесса. Спроектированный в Intel по 45 нм нормам транзистор примерно на 20 % опережает своего 65 нм собрата по скоростным характеристикам и оказывается примерно на 30 % экономичнее с точки зрения затрат энергии на переключение.

Часто вместо характеристики **энергопотребление** используют характеристику **рассеиваемая тепловая мощность** процессора. Для этого используется специальный термин **TDP**, который расшифровывается как **термопакет** (thermal design package) – это величина, показывающая, на отвод какой тепловой мощности должна быть рассчитана система охлаждения процессора.

Как правило, TDP показывает не максимальное теоретическое тепловыделение процессора, а типичное тепловыделение в реальных приложениях. Иногда, при длительных нагрузках на процессор (например, при кодировании видео), температура процессора может превысить заданный TDP. В этих случаях современные процессоры или дают сигнал выключения компьютера, или переходят в так называемый режим тротлинга (trottling), когда процессор пропускает часть тактов.

К другим технико-эксплуатационным характеристикам ЭВМ относятся:

- разрядность обрабатываемых слов и кодовых шин интерфейса;
- типы системного и локального интерфейсов;
- тип и емкость оперативной памяти;
- тип и емкость накопителя на жестком магнитном диске;
- тип и емкость кэш-памяти;
- тип видеоадаптера и видеомонитора;
- наличие средств для работы в компьютерной сети;
- наличие и тип программного обеспечения;
- надежность ЭВМ;
- стоимость;
- габариты и масса.

Производительность, пропускная способность, время отклика

Время отклика (Response time) – общее время, требующееся компьютеру для завершения задачи, включая время доступа к диску, памяти, активации устройств ввода-вывода, издержек операционной системы, времени выполнения задачи центральным процессором.

Пропускная способность (throughput, bandwidth) – еще один показатель производительности, представляющий собой количество задач, выполненных в единицу времени.

Для увеличения производительности требуется минимизировать время отклика или время выполнения какой-нибудь задачи. Таким образом, производительность и время выполнения для компьютера X связаны следующей формулой:

$$\text{Производительность}_X = 1/\text{Время выполнения}_X$$

Таким образом, для двух компьютеров X и Y, если производительность X выше, чем производительность Y, то получаем:

$$\text{Производительность}_X > \text{Производительность}_Y$$

$$1/\text{Время выполнения}_X > 1/\text{Время выполнения}_Y$$

$$\text{Время выполнения}_X < \text{Время выполнения}_Y$$

Чтобы упростить терминологию при количественном сравнении компьютеров, можно применять термин *быстрее чем*. Поскольку производительность и время выполнения являются обратными величинами, увеличение производительности требует уменьшения времени выполнения. Для того, чтобы избежать путаницы между терминами увеличе-

ние и уменьшение следует применять выражение «улучшение производительности» или «улучшение времени выполнения».

Оценка производительности

Оценочным критерием компьютерной производительности является время: компьютер, выполняющий тот же объём работы за меньшее время, является более быстрым. *Время выполнения* программ оценивается в секундах, затраченных на ее выполнение. Но время может быть определено и другими способами, в зависимости от того, что берется в расчёт. Наиболее простые – время выполнения процесса, время отклика или общее затраченное время. Эти понятия означают полное время, затраченное на выполнение задачи, включая обращения к диску, к памяти, работу устройств ввода-вывода, издержки, связанные с работой операционной системы.

Компьютеры часто работают в режиме разделения времени, а процессор может работать над выполнением сразу нескольких программ. В таких случаях система может стремиться не к минимизации общего времени, затраченного на выполнение одной программы, а к оптимизации пропускной способности.

Время выполнения задачи центральным процессором – это время, которое центральный процессор тратит на работу над этой задачей и включает время ожидания операций ввода-вывода или время затраченное на выполнение других программ.

Процессорное время пользователя – время центрального процессора, затраченное на саму программу.

Процессорное время системы – время центрального процессора, затраченное в операционной системе на выполнение задач, связанных с выполнением программ.

Производительность программ, цикл, такт

Для разных приложений особо важную роль играют разные аспекты производительности компьютерной системы. Многие приложения, особенно те, которые работают на серверах, сильно зависят от производительности устройств ввода–вывода, которая в свою очередь зависят как от аппаратного, так и от программного обеспечения. В некоторых условиях пользователя может интересовать пропускная способность, время отклика или сложная комбинация из этих двух показателей. Для повышения производительности программ нужно четко определить значения показатели производительности, а затем приступить к поиску

возможных узких мест и повышать производительность в различных частях системы.

Разработчики аппаратного обеспечения компьютеров могут анализировать компьютер, используя показатель, описывающий насколько быстро аппаратура может выполнять основные функции. Практически все компьютеры используют **тактовый генератор,** определяющий момент наступления событий. Эти дискретные интервалы времени называются **тактовыми циклами (или тактами).** Конструкторы ссылаются на продолжительность **тактового интервала** и как на *время завершения тактового цикла*, и как на *тактовую частоту*.

Производительность центрального процессора и ее факторы

Пользователи и разработчики часто оценивают производительность исходя из разных критериев. Если удастся соотнести эти разные критерии, то можно определить воздействие конструктивных изменений на производительность в понятиях пользователя. Если рассматривать производительность процессора, то основные показатели процессорного времени (количество тактовых циклов и время цикла тактового генератора) связаны формулой:

Время выполнения программы ЦП	=	Кол-во тактовых циклов процессора, затраченных на программу	x Время цикла тактового генератора

Если время цикла определить как величину обратную частоте, то получим:

$$\text{Время выполнения} = \frac{\text{Кол} - \text{во тактовых циклов процессора}}{\text{Тактовая частота}}$$

Таким образом, увеличить производительность можно сократив **количество тактовых циклов** или сократив **продолжительность тактового цикла**.

Задача

Интересующая нас программа выполняется на компьютере А с тактовой частотой 2Ггц за 10 сек. Необходимо создать компьютер, который выполнял бы эту программу за 6 сек. Разработчик определил, что соответствующее повышение тактовой частоты вполне допустимо, но это повышение повлияет на всю остальную конструкцию центрального процессора и компьютеру В на выполнение программы потребуется в

1,2 раза больше тактовых циклов, чем компьютеру А. Какую тактовую частоту следует заказать разработчику компьютеров.

Решение

$$\text{Процессорное время}_A = \frac{\text{Количество тактовых циклов процессора}_A}{\text{Тактовая частота}_A}$$

$$10 \text{ секунд} = \frac{\text{Количество тактовых циклов процессора}_A}{2 \times 10^9 \text{ циклов в секунду}}$$

$$\text{Количество тактовых циклов процессора}_A = 10 \text{ секунд} \times 2 \times 10^9 \text{ циклов в секунду} = 20 \times 10^9 \text{ циклов}$$

Процессорное время для компьютера В может быть найдено с использованием следующего уравнения:

$$\text{Процессорное время}_B = \frac{1,2 \times 20 \times \text{Количество тактовых циклов процессора}_A}{\text{Тактовая частота}_B}$$

$$6 \text{ секунд} = \frac{1,2 \times 20 \times 10^9 \text{ циклов}}{\text{Тактовая частота}_B}$$

$$\text{Тактовая частота}_B = \frac{1,2 \times 20 \times 10^9 \text{ циклов}}{6 \text{ секунд}} = 1,2 \times 20 \times 10^9 \text{ циклов в секунду} = 4 \times 10^9 \text{ циклов в секунду} = 4\text{ГГц}$$

Таким образом, чтобы выполнить программу за 6 сек., компьютер В должен иметь вдвое больше частоту, чем компьютер А.

Показанные выше уравнения не учитывают количество инструкций, необходимое для выполнения программ. Одним из способов представления времени выполнения состоит в том, что оно равно количеству выполненных инструкций, умноженному на среднее время выполнения одной инструкции. Таким образом, количество тактовых циклов, требуемое для программы, может быть записано в виде следующей формулы:

Кол-во циклов	тактовых	=	Кол-во инструкций для программы	x	Среднее кол-во тактовых циклов на инструкцию

Для термина **количество тактовых циклов на инструкцию**, означающего среднее количество тактовых циклов, затрачиваемых на выполнение каждой инструкции, часто используется сокращение **CPI** (**clock cycles per instruction**). Поскольку на разные инструкции, в зависимости от того, что именно они делают, тратится разное количество времени, CPI является усредненным показателем для всех инструкций, выполняемых в программе. CPI предоставляет единственный способ сравнения двух различных реализаций одной и той же архитектуры набора инструкций, поскольку количество инструкций, выполняемых в программе, будет одинаковым.

$$CPI = \frac{\text{Количество тактовых циклов процессора}}{\text{Количество инструкций}}.$$

Классическое уравнение производительности ЦП можно записать в следующем виде:

$$\text{Процессорное время} = \text{Количество инструкций} \times \text{CPI} \times \text{Продолжительность тактового цикла}$$

либо

$$\text{Процессорное время} = \frac{\text{Количество инструкций} \times CPI}{\text{Тактовая частота}}.$$

Задача

Предположим, что есть две реализации одной и той же архитектуры набора команд. У компьютера А продолжительность тактового цикла равна 250 пс, значение CPI для некой программы равно 2.0, а у компьютера В продолжительность тактового цикла равна 500 пс. и значения CPI равно 1,2. Какой из компьютеров будет быстрее для данной программы и насколько.

Решение

Мы знаем, что любой компьютер при каждом запуске программы выполняет одно и то же количество инструкций; обозначим это количество буквой *I*. Сначала определим количество тактовых циклов процессора для каждого компьютера:

$$\text{Количество тактовых циклов процессора}_A = I \times 2{,}0$$

$$\text{Количество тактовых циклов процессора}_B = I \times 1{,}2.$$

Теперь мы можем вычислить процессорное время для каждого компьютера:

$$\text{Процессорное время}_A = \text{Количество тактовых циклов процессора}_A \times \text{продолжительность тактового цикла} = I \times 2{,}0 \times 250 \text{ пс} = 500 \times I\text{пс}.$$

И точно так же для В:

$$\text{Процессорное время}_B \times I \times 1{,}2 \times 500 \text{ пс} = 600 \ I\text{пс}.$$

Совершенно очевидно, что компьютер А работает быстрее. Насколько быстрее, позволяет определить соотношение показателей времени выполнения:

$$\frac{\text{Производительность процессора}_A}{\text{Производительность процессора}_B} = \frac{\text{Время выполнения}_B}{\text{Время выполнения}_B} = \frac{600 \times I\text{пс}}{500 \times I\text{пс}} = 1{,}2.$$

Всегда нужно помнить, что единственным полноценным и достоверным оценочным показателем производительности компьютера является **время**. Например, изменение набора команд, направленное на

снижение количества инструкций, может привести к созданию конструкции с более медленным тактовым циклом или с более высоким показателем CPI, что нивелирует все преимущества от уменьшения количества инструкций. Аналогично этому, поскольку показатель CPI зависит от типа выполняемых инструкций, код, который выполняется меньшее количество инструкций, может быть далеко не самым быстрым.

В таблице 1.3.1 приведены основные составляющие производительности и способы их оценки

Таблица 1.3.1. Основные составляющие производительности и способы их оценки

Составляющая производительности	**В чем измеряется**
Время выполнения программы центральным процессором	В секундах на программу
Количество инструкций	В инструкциях, использованных для выполнения программы
Количество тактовых циклов на инструкции (CPI)	В среднем количестве тактовых циклов на одну инструкцию
Продолжительность тактового цикла	В секундах на тактовый цикл

Время выполнения программы ЦП можно оценить, запустив программу, а продолжительность тактового цикла обычно приводится в документации компьютера. CPI определяется сложнее, но если известна тактовая частота и время выполнения программы ЦП, то для определения всего остального нужно получить лишь количество инструкций или CPI.

Количество инструкций можно определить, используя программные средства, предназначенные для исследования процесса выполнения или эмулятор архитектуры. Кроме этого, для записи различных оценочных показателей CPI, и для определения источников потери производительности можно воспользоваться аппаратными счетчиками, имеющимися у большинства процессоров. Поскольку количество инструкций зависит от архитектуры, но не от ее конкретной реализации, его можно определить и без знания всех подробностей реализации. А вот CPI зависит от разнообразных конструктивных особенностей компьютера, включая и систему памяти и структуру процессора, а также от сочета-

ния типов инструкций, выполняемых по программе (набор инструкций). Таким образом, показатель CPI варьируется от приложения к приложению и от реализации к реализации одного и того же набора инструкций.

При сравнении двух компьютеров нужно принимать во внимание все три компонента, которые обусловливают время выполнения. Если некоторые факторы имеют одинаковое значение, производительность может быть определена сравнением всех отличающихся по значению факторов.

Представление о производительности программ

Производительность программ зависит от алгоритма, языка, компилятора, архитектуры и используемого аппаратного обеспечения (таблица 1.3.2)

Таблица 1.3.2. Влияние компонентов компьютера на производительность

Аппаратный или программный компонент	**На что влияет**	**Как влияет**
Алгоритм	На количество инструкций и возможно на CPI	Алгоритм определяет количество инструкций для выполнения исходной программы, а следовательно, и количество выполняемых инструкций. Алгоритм может также влиять на CPI за счет предпочтения более медленных или более быстрых инструкций. Например, если алгоритм использует больше операций с плавающей точкой, это может привести к более высокому CPI.
Язык программирования	На количество инструкций и на CPI	Язык программирования влияет на количество инструкций, поскольку инструкции языка транслируются в инструкции процессора. Язык, в силу своих свойств, может также влиять на CPI, например, язык с мощной поддержкой абстракций данных (к примеру, Java) может потребовать применение вызовов, использующих более затратные инструкции.

Аппаратный или программный компонент	На что влияет	Как влияет
Компилятор	На количество инструкций и на CPI	Эффективность работы компилятора влияет и на количество инструкций, и на среднее количество тактов, приходящееся на одну инструкцию, поскольку компилятор определяет ход трансляции инструкций языка в инструкции процессора. Компилятор может играть весьма сложную роль и осуществлять комплексное воздействие на CPI
Архитектура набора команд	На количество инструкций, на тактовую частоту, на CPI	Архитектура набора команд воздействует на все 3 аспекта производительности ЦП, поскольку она влияет на инструкции, необходимые для выполнения функций, на количество циклов, необходимое для выполнения каждой инструкции, и, в общем, на тактовую частоту процессора

.

2 ФУНКЦИОНАЛЬНАЯ И СТРУКТУРНАЯ ОРГАНИЗАЦИЯ ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

2.1 Производство интегральных микросхем

Производство интегральных микросхем и чипов базируется на **кремнии**. Поскольку кремний не является хорошим проводником электрического тока, его называют **полупроводником**. С помощью специального химического процесса кремний можно трансформировать:

- в отличные проводники электрического тока (при помощи использования микроскопических медных или алюминиевых проводков)
- в отличные изоляторы, не пропускающие электрический ток (подобные пластмассовым или стеклянным изоляторам);
- в блоки, которые при определенных условиях являются либо проводниками, либо изоляторами (как переключатели);

Транзисторы – полупроводниковые элементы с тремя выводами (обычно), на один из которых (коллектор) подаётся сильный ток, а на другой (база) подаётся слабый (управляющий ток) относятся к последней из описанных выше преобразований кремния.

Сверхбольшие интегральные схемы (СБИС или VLSI) состоят из миллиардов комбинаций проводников, изоляторов и переключателей, изготавливаемых в одном небольшом корпусе.

Процесс производства интегральных микросхем существенным образом сказывается на стоимости чипов, что соответственно важно для разработчиков компьютеров. На рис. 2.1.1 представлен процесс производства микросхем и чипов. Процесс начинается с выращенного кристалла кремния, по форме напоминающего огромную сосиску. Современные выращенные кристаллы имеют размеры 812 дюймов (20-30 см) в диаметре и 12–14 дюйма (30–60 см) в длину. Кристалл в конечном итоге разрезается на тонкие пластины (**вафли**) толщиной не более 0,1 дюйма. Затем эти вафли проходят несколько этапов обработки, во время которых на каждую вафлю накладываются шаблоны, создающие транзисторы, проводники и изоляторы.

Единственный микроскопический изъян в самой вафле или сбой в проведении одного из нескольких десятков этапов обработки может привести к выходу из строя участка вафли. Такие **дефекты** практически исключают возможность производства идеальной вафли. Чтобы справиться с дефектом, используется несколько приёмов, но самый простой

из них – помещение на вафлю множества независимых компонентов. Обработанная вафля затем нарезается на эти компоненты – заготовки под интегральную схему или **пластины,** а также называемые **чипы**. На рис. 2.1.2. приведена фотография вафли, содержащая микропроцессоры до их нарезки.

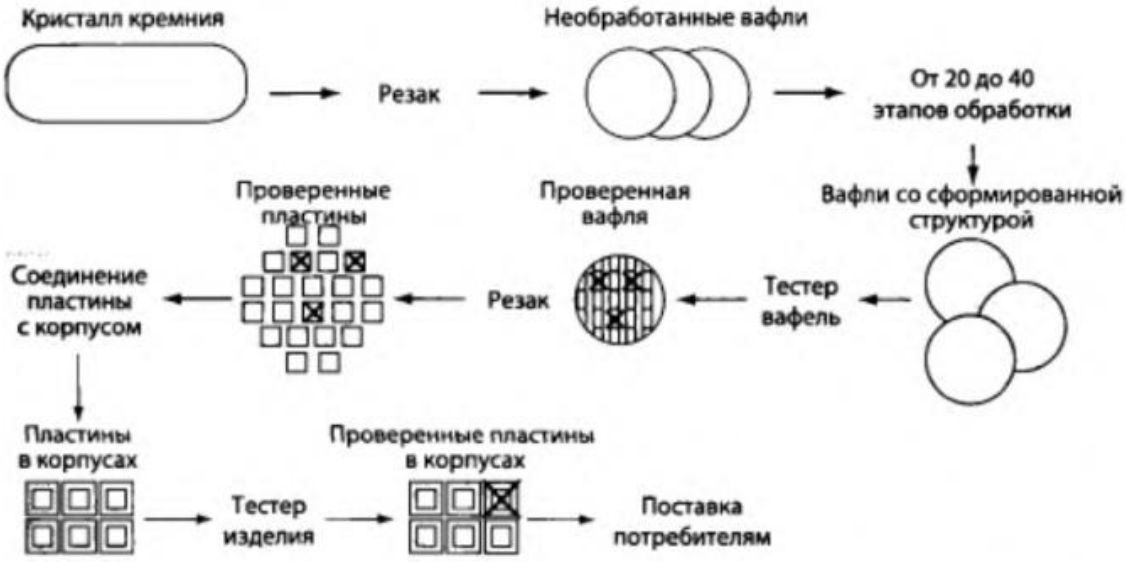


Рис. 2.1.1 Процесс производства чипов

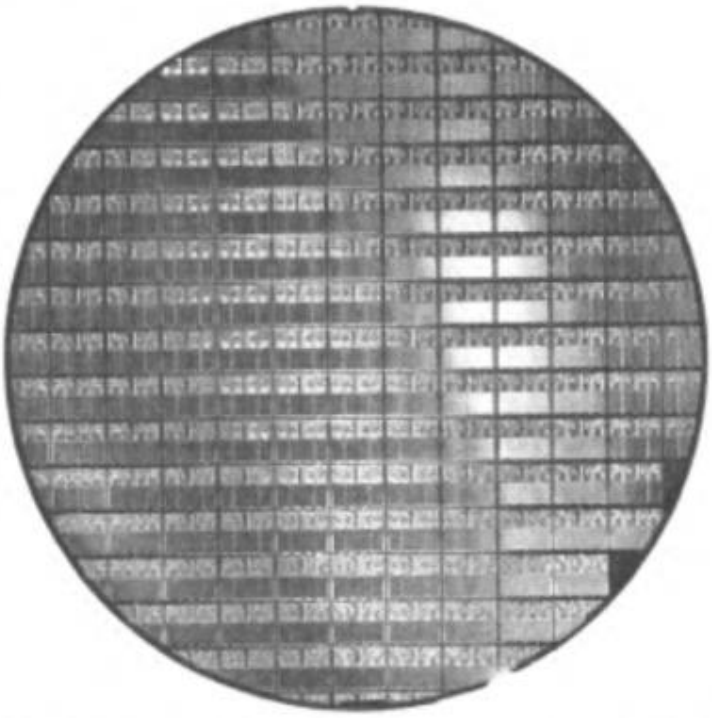


Рис. 2.1.2. Вафля, состоящая из чипов перед нарезкой

Нарезка позволяет отбраковать не всю вафлю, а только пластины с изъянами. Этот подход количественно определяется выходом годных изделий, который измеряется в процентном отношении годных пластин к общему числу пластин в вафле.

С увеличением размеров пластин быстро растёт стоимость интегральных микросхем, что обуславливается как меньшим выходом годных изделий, так и меньшим количеством пластин, помещающихся в вафле. Чтобы снизить стоимость, крупные пластины «сужаются» за счет более миниатюрных размеров транзисторов. При этом минимальный размер транзистора определяется **нормой технологического процесса производства полупроводниковых кристаллов.** На рис. 2.3. приведена вафля, произведенная по 90-нанометровой технологии, т.е самые маленькие транзисторы имеют размер около 90 нм.

В процессе сборки годные пластины соединяются с входными и выходными выводами корпуса. Готовые изделия проходят окончательную проверку, т.к. в процессе заключения в корпус могут допускаться ошибки.

Наиболее важным конструктивным ограничением является потребляемой мощность. Проблема мощности имеет 2 аспекта:

- мощность должна быть подведена к чипу и распределена по нему; современные микропроцессоры имеют сотни контактов для питания и заземления.
- Мощность рассеивается в виде тепла и должна быть отведена.

Стоимость интегральной микросхемы может быть выражена 3-мя простыми выражениями:

$$\text{Стоимость одной пластины} = \frac{\text{Стоимость вафли}}{\text{Количество пластин на вафле} \times \text{Коэффициент выхода}}$$

$$\text{Количество пластин на вафле} = \frac{\text{Площадь вафли}}{\text{Площадь пластины}}$$

$$\text{Выход} = \frac{1}{(1 + (\text{Дефектов на площадь} \times \text{Площадь пластины}/2))^2}$$

2.2 Назначение и структура центрального процессора

Центральный процессор – основное устройство ЭВМ, которое наряду с обработкой данных выполняет функции управления системой: инициирование ввода/вывода, обработку системных событий, управление доступом к основной памяти и т.п.

Структурная организация центрального процессора (ЦП) определяется функционально-логической организацией, микроархитектурой и требованиями к технико-экономическим показателям.

Логическую структуру ЦП представляет ряд функциональных средств (рис. 2.2.1).: средства обработки, средства управления системой и программой (центральное устройство управления), локальная память, буферная память (кэш-память L1, L2, …), средства инициализации ввода/вывода, средства управления памятью, системные средства.

Средства обработки обеспечивают выполнение операций над данными с фиксированной (целочисленные данные) и плавающей точкой, векторными данными, полями переменной длины и т.д. Локальная память состоит из регистров общего назначения, регистров данных с плавающей точкой, векторных регистров, управляющих регистров и др. К средствам управления памятью относятся средства управления доступом к ОП и предвыборкой команд и данных. Буферная память включает в себя кэш-память команд и данных первого (L1), второго (L2) и третьего (L3) уровней. Средства инициализации ввода/вывода обеспечивают активизацию контроллеров (каналов) периферийных устройств. К системным средствам относятся средства службы времени: часы астрономического времени, таймер, коммутатор и т.д. Существует обязательный (стандартный) минимальный набор функциональных средств для каждого типа центрального процессора. Он включает в себя регистры общего назначения, средства выполнения стандартного набора операций и средства управления вычислительным процессом.

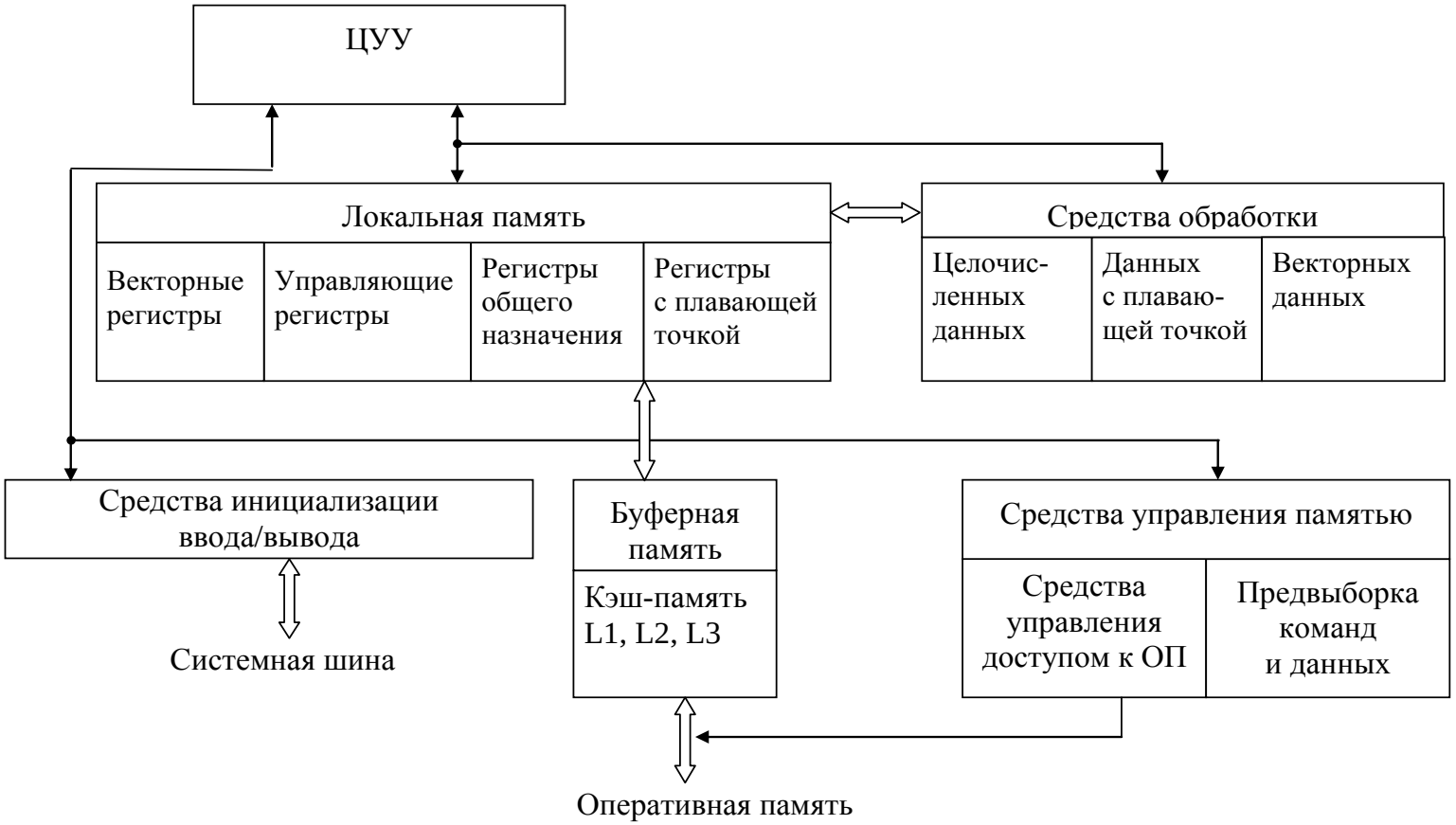


Рис. 2.2.1 Логическая организация ЦП

2.3 Архитектура процессора

Существует 2 подхода к толкованию значения термина «Архитектура процессора»:

С точки зрения программистов, под архитектурой процессора подразумевается его способность исполнять определенный набор машинных команд (инструкций). Большинство современных десктопных CPU относятся к семейству x86, или Intel-совместимых процессоров архитектуры IA-32 (архитектура 32-битных процессоров Intel).

С точки зрения разработчиков компьютерного железа архитектура процессора отражает основные принципы **внутренней организации конкретных семейств процессоров** или некий набор свойств и качеств, присущий целому семейству процессоров.

Исторически первыми появились однопроцессорные архитектуры. Классическим примером однопроцессорной архитектуры является архитектура **фон Неймана** со строго последовательным выполнением команд: процессор по очереди выбирает команды программы и также по очереди обрабатывает данные. В архитектуре фон Неймана применяется однородная память микропроцессора. В эту память могут записываться различные программы. При этом специальная программа-загрузчик работает с ними как с данными. Затем управление может быть передано этим программам и они уже начинают выполнять свой алгоритм. При подобном подходе к управлению микропроцессором удается достигнуть максимальной гибкости микропроцессорной системы. По мере развития вычислительной техники архитектура фон Неймана обогатилась сначала конвейером команд (подробнее конвейер рассмотрен в разделе 2.4), а затем многофункциональной обработкой (рис. 2.3.1) и получила обобщенное название **SISD (Single Instruction Single Data – один поток команд, один поток данных)**.

Рис. 2.3.1. Развитие и классификация однопроцессорных архитектур

Существует также второй тип архитектуры, менее популярный, но также применяемый в процессорах – **Гарвардская** архитектура. В Гарвардской архитектуре принципиально различаются два вида памяти микропроцессора, которые разделены физически:

- память программ (для хранения инструкций микропроцессора)
- память данных (для временного хранения и обработки переменных)

В гарвардской архитектуре принципиально невозможно осуществить операцию записи в память программ, что исключает возможность случайного разрушения управляющей программы в случае ошибки программы при работе с данными или атаки третьих лиц. Кроме того, для работы с памятью программ и с памятью данных организуются отдельные шины обмена данными (системные шины), как это показано на структурной схеме, приведенной на рисунке 2.3.2

Гарвардская архитектура применяется в микроконтроллерах и в сигнальных процессорах, где требуется обеспечить высокую надёжность работы аппаратуры.

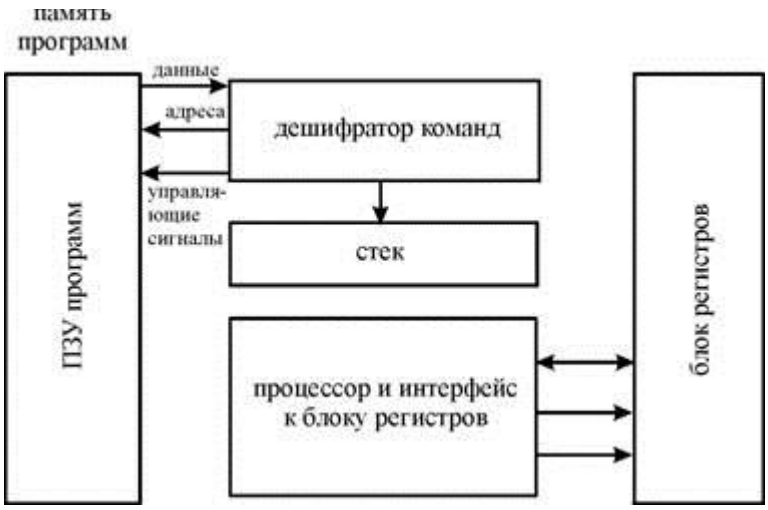


Рис. 2.3.2. Структурная схема гарвардской архитектуры

Архитектура SISD породила целый ряд архитектур: CISC, RISC, VLIW и EPIC-концепцию.

CISC-архитектура

Компьютеры с CISC (Complex Instruction Set Computer)-архитектурой имеют комплексную (полную) систему команд, под управлением которой выполняются всевозможные операции типа «память – память», «память – регистр», «регистр – память», «регистр – регистр».

CISC-архитектура появилась еще на заре вычислительной техники. Лидером в разработке микропроцессоров с полным набором команд считается компания Intel со своей серией процессоров x86, Pentium, Intel Core и др. Эта архитектура, получившая название x86, является практически стандартом на рынке микропроцессоров.

Данная архитектура характеризуется:

- большим числом команд (более 200);
- переменной длиной команд (от 1 до 13 байт);
- значительным числом способов адресации и форматов команд;
- наличием сложных команд и многотактностью их выполнения;
- наличием микропрограммного управления для сложных команд.

На мировых рынках полная система команд x86 представлена в процессорах фирм Intel, AMD, VIA Technologies и др.

RISC-архитектура

Компьютеры с RISC (Reduced Instruction Set Computer)-архитектурой содержат набор простых, часто употребляемых в программах команд. Основными являются операции типа «регистр – регистр».

Понятие RISC в современном его понимании оформилось на базе трех исследовательских проектов компьютеров: процессора 801 компании IBM, процессора RISC университета Беркли и процессора MIPS Стенфордского университета. Простота архитектуры и ее эффективность, подтвержденная этими проектами, вызвали большой интерес в компьютерной индустрии, и с 1986 г. началась активная промышленная реализация архитектуры RISC. Отличительные черты данной архитектуры:

- сокращенное число команд;
- большинство команд выполняется за один машинный такт;
- постоянная длина команд;
- небольшое количество способов адресации и форматов команд;
- для простых команд нет необходимости в использовании микропрограммного управления;
- большое число регистров внутренней памяти процессора.

Исходя из перечисленных характеристик, компьютеры с RISC-архитектурой «обязаны» иметь преимущество в производительности по сравнению с CISC-компьютерами.

В начале 90-х ярким представителем архитектуры RISC были процессоры PowerPC. В настоящее время основными разработчиками RISC-процессоров являются корпорации Sun/Oracle (Ultra Sparc T1, T2), IBM (POWER 6, 6+, 7, 8, Cell). Эти процессоры используются в высокопроизводительных компьютерах (рабочих станциях, серверах, суперкомпьютерах).

Для мобильных устройств (карманных ПК, смартфонов, коммуникаторов) наибольшее распространение получили RISC-процессоры семейства ARM (корпорация ARM Ltd, Великобритания).

Уступая во многом RISC, процессоры с системой команд x86 сохранили лидерство на рынке персональных систем за счет постоянной модернизации системы команд, нацеленной на увеличение производительности процессоров, а также за счет того, что программное обеспечение, разработанное для x86-компьютеров, начиная с 1980 г., способно функционировать и на современных компьютерах с этой архитектурой. В свою очередь, достоинства RISC-процессоров укрепили их позиции на более молодом рынке высокопроизводительных машин (рабочих станций, серверов).

В начале 90-х гг. между представителями этих архитектур началась острая конкуренция за превентивное улучшение характеристик, в первую очередь производительности и ее отношения к трудоемкости разработки процессоров. Создатели CISC- и RISC-процессоров нередко боролись с конкурентами, заимствуя их удачные решения. Например, компания Intel реализовала в процессоре Pentium Pro (шестое поколение P6 процессоров Intel) RISC-подобную организацию вычислений. В P6 изощренно построенный декодер транслирует сложные команды x86 в более короткие и простые RISC-микрокоманды. В архитектуре P6 RISC-решения впервые в семействе x86 перестали быть лишь дополнением исконных CISC-средств повышения производительности. Поэтому частица Pro в названии первого процессора этой серии обозначает «Полноценная RISC-архитектура» (Precision RISC Organization).

VLIW-архитектура

VLIW-архитектура связана с кардинальной перестройкой всего процесса трансляции и исполнения программ. Уже на этапе подготовки программы компилятор группирует несвязанные операции в пакеты, содержимое которых строго соответствует структуре процессора. Сформированные пакеты операций преобразуются компилятором в командные слова, которые по сравнению с обычными инструкциями выглядят очень большими. Отсюда и название этих суперкоманд, и соответствующей им архитектуры – VLIW (Very Long Instruction Word – очень длинное командное слово). По идее, затраты на формирование суперкоманд должны окупаться скоростью их выполнения и простотой аппаратуры процессора, с которого снята вся «интеллектуальная» работа по поиску параллелизма несвязанных операций.

Компилятор VLIW, в отличие от суперскалярной обработки, производит статический анализ программы и создает точный план того, как процессор будет выполнять программу: указывается, когда будет выполнена каждая операция, какие функциональные устройства будут работать и какие регистры будут содержать операнды.

Компилятор VLIW передает план вычисления аппаратному обеспечению, которое, в свою очередь, выполняет указанный план. Этот план позволяет VLIW использовать относительно простое аппаратное обеспечение, способное добиться высокого уровня параллелизма на уровне команд.

Однако даже при небольшом изменении начальных данных путь выполнения программы сколь угодно сильно изменяется.

VLIW-архитектура в свое время использовалась в RISC-процессорах семейств PA-8000, 9000 корпорации HP (Hewlett Packard), сейчас при-

меняется в отечественных процессорах семейства «Эльбрус», а также в процессорах цифровой обработки сигналов различных фирм производителей.

Аббревиатуры рассмотренных архитектур CISC, RISC, VLIW в настоящее время обозначают только идеализированные концепции. Реальные микропроцессоры трудно классифицировать. Современные микропроцессоры, причисляемые к RISC, сильно отличаются от первых процессоров RISC-архитектуры. То же относится и к CISC. Просто в наиболее совершенных процессорах заложено множество удачных идей, вне зависимости от их принадлежности к какой-либо архитектуре.

Концепция EPIC

Концепция EPIC (Explicity Parallel Instruction Computing – вычисления с явным параллелизмом команд, где «явным» означает явно указанным при трансляции) разработана совместно фирмами Intel и Hewlett Packard и имеет ту же значимость, что и CISC- и RISC-архитектуры.

Концепция реализации параллелизма на уровне команд (EPIC) определяет новый тип архитектуры, способной конкурировать по масштабам влияния с RISC. Эта идеология направлена на то, чтобы упростить аппаратное обеспечение и, в то же время, извлечь как можно больше «скрытого параллелизма» на уровне команд, чем это можно сделать при реализации VLIW и суперскалярных стратегий, используя большую ширину «выдачи» команд и длинные (глубокие) конвейеры.

Одна из целей, которые ставили перед собой разработчики при создании EPIC, состояла в том, чтобы сохранить реализованный во VLIW принцип статического создания плана вычислений, но, в то же время, обогатить его возможностями, аналогичными возможностям суперскалярного процессора, позволяющими новой архитектуре лучше учитывать динамические факторы, традиционно ограничивающие параллелизм, свойственный VLIW. EPIC предоставляет динамические механизмы на уровне аппаратуры так, что компилятор может управлять такими средствами, применяя их выборочно, где это возможно. Столь широкие возможности помогают компилятору использовать правила управления этими механизмами более оптимально, чем это позволяет аппаратура.

Концепция EPIC, согласно Intel и HP, обладает достоинствами VLIW, но не обладает ее недостатками.

EPIC-технология с явным заимствованием лучших идей из CISC- и RISC-архитектур использована в 64-разрядной интеловской архитектуре (IA-64) процессоров Itanium, Itanium2.

Особенности IA-64:

- использование новой системы команд, разработанный Intel и HP;
- большое количество регистров (128 64-разрядных регистров общего назначения);
- использование простых инструкций, сгруппированных по три, одинаковой длины, образующих длинные командные слова LIW (long instruction words);
- переупорядочиванием и оптимизацией команд, так же как и во VLIW, занимается компилятор, а не процессор;
- команды из разных ветвей узлового ветвления снабжаются предикатными полями (полями условий) и запускаются параллельно;
- выборка данных по предположению (выборка данных до того, как они потребуются, т.е. заранее);
- масштабируемость архитектуры до большого количества функциональных устройств.

Процессор Itanium не только реализует новые возможности 64-разрядной архитектуры, но и обладает аппаратной совместимостью с набором команд IA-32.

SIMD-архитектура

Параллелизм циклов и итераций тесно связан с понятием множественности потоков данных и реализуется векторной обработкой. В классификации компьютерных архитектур выделена специальная группа однопроцессорных систем с параллельной обработкой потоков данных – **SIMD (Single Instruction Multiple Data**, один поток команд – множество потоков данных). Существуют несколько способов (рис. 2.3.3) реализации этой архитектуры: матричная структура процессора, векторно-конвейерная, технология MMX и потоковые SIMD-расширения в интеловских процессорах.

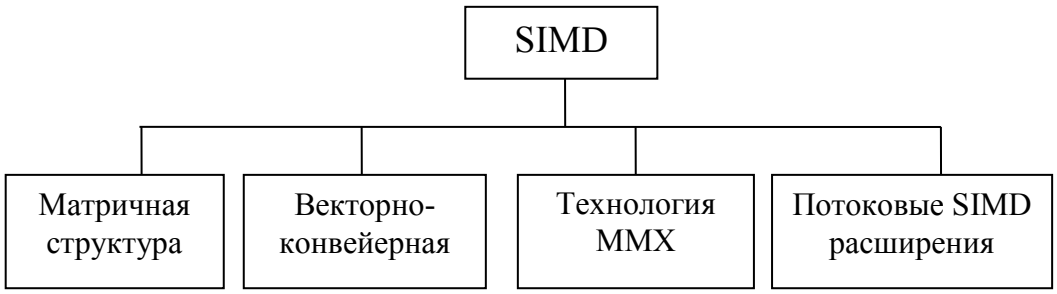


Рис. 2.3.3. Классификация способов организации SIMD-архитектуры

Суть **матричной структуры** заключается в том, что имеется множество процессорных элементов, исполняющих одну и ту же команду

над различными элементами матрицы, объединенных коммутатором. Основная проблема заключается в программировании обмена данными между процессорными элементами через коммутатор.

В отличие от матричной **векторно-конвейерная структура** процессора содержит конвейер операций, на котором обрабатываются параллельно элементы векторов и полученные результаты последовательно записываются в единую память. При этом отпадает необходимость в коммутаторе процессорных элементов, служащем камнем преткновения в матричных компьютерах.

Общим для всех векторных компьютеров является наличие в системе команд векторных операций, допускающих работу с векторами определенной длины. В таких компьютерах операции с векторами обычно выполняются над содержимым векторных регистров.

Одним из примеров реализации SIMD-архитектуры является **технология MMX**, которая существенно улучшила архитектуру микропроцессоров фирмы Intel (Pentium MMX). Она разработана для ускорения выполнения мультимедийных и коммуникационных программ. Команды MMX выполняют одну и ту же функцию с различными частями данных, например, 8 байт графических данных передаются в процессор как одно упакованное 64-разрядное число и обрабатываются одной командой.

Следующим шагом по пути использования SIMD-архитектуры в микропроцессорах фирмы Intel (Pentium III) явились **потоковые SIMD-расширения – Streaming SIMD Extension (SSE),** которые реализуют новые SIMD-инструкции, оперирующие со специальными 128-битными регистрами (подробнее о регистрах SSE описано в главе 2.7). Каждый из этих регистров может хранить несколько упакованных целочисленных или вещественных данных (подробнее о структуре данных SSE описано в разделе 2.4). Таким образом, выполняя операцию над содержимым двух регистров под управлением команды SSE, процессор может обработать несколько пар операндов одновременно.

Несколько раньше то же самое было сделано фирмой AMD – расширение 3DNow!, которое было реализовано уже в процессорах K6-2 с введением новых инструкций, оперирующих с 64-битными регистрами.

Данное направление получило развитие и в следующих поколениях процессоров корпораций Intel и AMD. Современные процессоры Intel поддерживают потоковые расширения SSE, SSE2, SSE3, SSSE3, SSE4, AVX, AVX2, MIC и т.д.

2.4 Конвейерная обработка команд

2.4.1 Общее понятие конвейера

Конвейеризация – это техника реализации, в которой выполнение нескольких инструкций осуществляется одновременно. В современных процессорах конвейер используется повсеместно и является основным способом обработки команд.

В данном разделе рассмотрен конвейер на примере прачечной. Если процесс стирки разбить на этапы, то они будут примерно следующими:

1. Загрузить стиральную машину одной порцией белья.
2. Когда стиральная машина завершит стирку, поместить мокрое белье в сушилку.
3. Когда сушилка завершит работу, положить высушенное белье на стол и сложить его.
4. Когда белье будет сложено – унести его.

Когда белье будет унесено, повторить все это с новой партией белья.

Подход, использующий конвейеризацию, как показано на рис. 2.4.1., занимает намного меньше времени.

Процедура выполнения команд процессором включает несколько характерных этапов. В простейшем случае можно выделить, как минимум, четыре этапа обработки команд (рис. 2.4.2, *а*): выборка команды (ВК), декодирование команды (ДК), выполнение операции (ОП) и запись результата (ЗР).

Каждый этап в процессоре выполняется за один такт. При последовательной обработке команд (рис. 2.4.2, *б*), выполнение следующей (n + 1)-й команды начинается только после завершения предыдущей (n)-й команды. Это приводит к низкой производительности и простоям аппаратуры процессора.

Для улучшения этих характеристик используется параллельное выполнение нескольких команд путем совмещения в каждом такте различных этапов их обработки (рис. 2.4.2, *в*). После выборки *n-й* команды во 2-м такте идет ее декодирование и выборка (n + 1)-й команды. В третьем такте выполняется n-я команда, декодируется (n + 2)-я и осуществляется выборка (n + 3)-й команды и т.д. Такая организация работы процессора называется **конвейерной обработкой (конвейером команд).**

Совмещенные принципы обработки (конвейер команд) существенно увеличивают пропускную способность процессора.

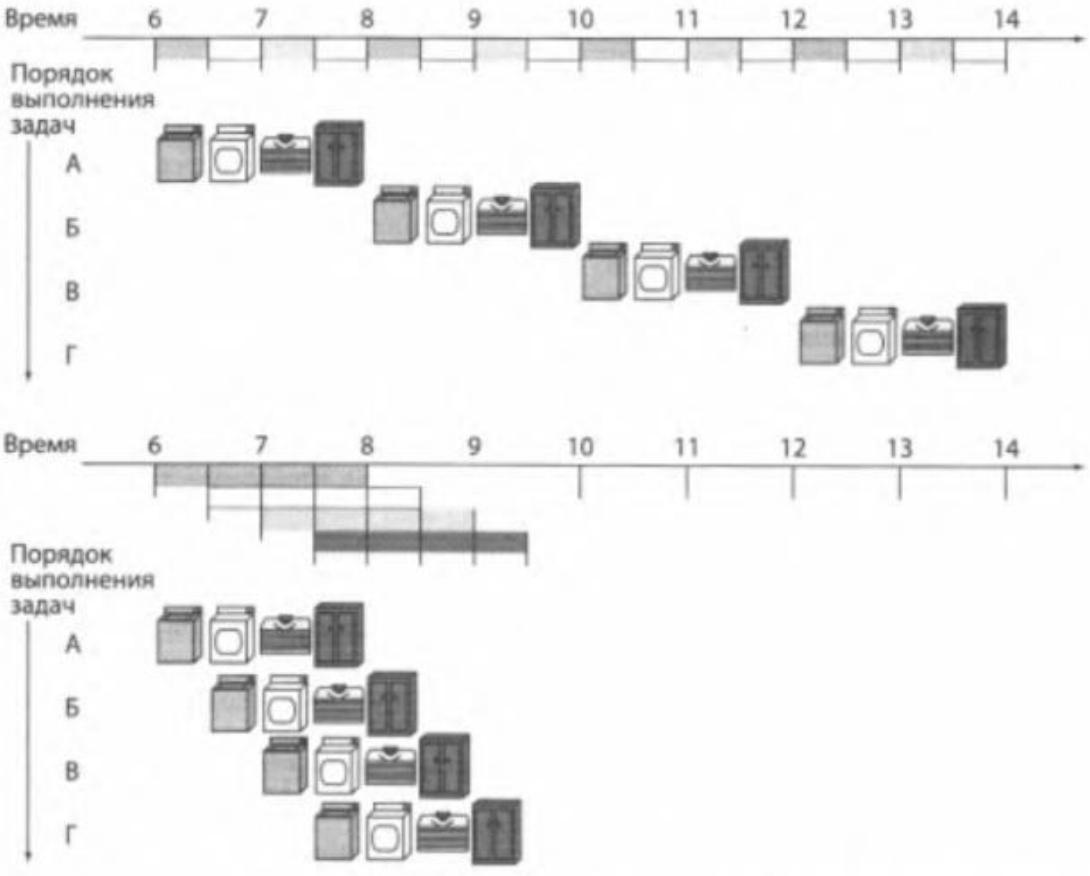


Рис. 2.4.1. Конвейер на примере прачечной

Приостановку работы конвейера вызывает любая команда **условного перехода** в программе или взаимозависимость команд, т.е. использование следующей командой результатов предыдущей команды.

Конечно, рассмотренный процессор является гипотетическим. В реальных процессорах конвейер обработки команд сложнее и включает большее количество ступеней. Причина увеличения длины конвейера заключается в том, что многие команды являются довольно сложными и не могут быть выполнены за один такт процессора, особенно при высоких тактовых частотах. Поэтому каждая из четырех стадий обработки команд (выборка, декодирование, выполнение и запись) может состоять из нескольких ступеней конвейера. Собственно, длина конвейера – это одна из наиболее значимых характеристик любого процессора. Чем больше длина конвейера, тем большую частоту можно использовать в процессоре.

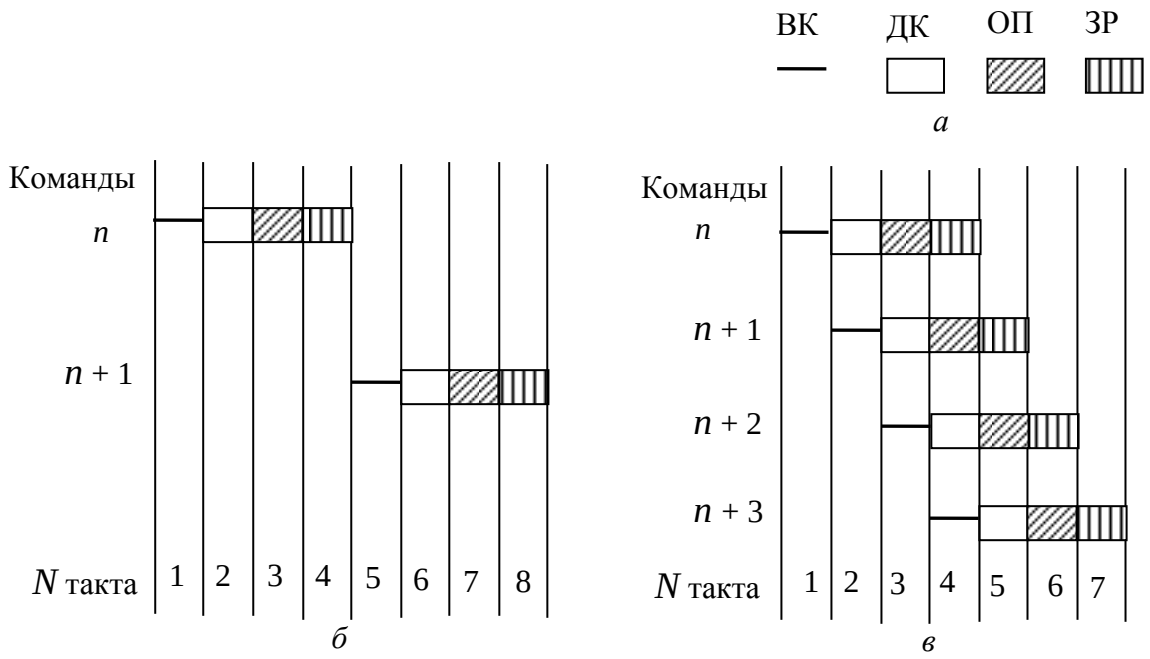


Рис. 2.4.2. Временные диаграммы обработки команд в процессоре:
а – этапы выполнения команды; *б* – последовательное выполнение команд;
в – совмещенное выполнение команд (конвейеризация)

Для обеспечения непрерывности вычислительного процесса в структуре ЦП используется блок прогнозирования переходов и устройство выполнения переходов.

2.4.2 Предсказание условных переходов

Условный переход возникает в случае наличия соответствующей инструкции, полученной в результате реализации операторов if else. Для обработки условных переходов процессоры используют принцип **прогнозирования**. Один из простых подходов заключается в прогнозировании того, что переход не состоится. Когда прогноз сбывается, конвейер работает на полной скорости и его задержку вызывают только состоявшиеся переходы.

Более сложная версия **прогнозирования условного перехода** предусматривает прогнозирование выполнения одних переходов и невыполнения других. Такие жесткие подходы к прогнозированию условного перехода основываются на стереотипе поведения и не рассчитаны на индивидуальный характер отдельной инструкции условного перехода. Абсолютной противоположностью являются **динамические** аппаратные средства прогнозирования, которые строят свой прогноз в

зависимости от поведения каждого условного перехода и способны изменять прогноз на переход за время жизни программы.

Одним из популярных подходов к динамическому прогнозированию условных переходов является ведение истории для каждого состоявшегося и несостоявшегося условного перехода, а затем использования недавнего поведения из прошлого для предсказания будущего. Как будет показано далее, рост количества и типа хронологических данных отразится на результате динамических прогнозов условного перехода, позволяя достичь точности прогноза более 90%. Когда прогноз не подтвердится, система управление конвейером должна сделать так, чтобы инструкция, следующая непосредственно за неверно спрогнозированной инструкцией условного перехода, не повлияла на дальнейший ход событий, и обеспечить перезапуск конвейера с подходящего адреса условного перехода. В нашей аналогии с прачечной нужно остановить загрузку новых партий, чтобы можно было перезапустить обработку новой партии, для которой была неверно спрогнозирована формула.

Как и в случае всех других способов разрешения конфликтов управления, более длинные конвейеры усложняют проблему, повышая, таким образом, ущерб от неверного прогнозирования.

Существует еще и упомянутый выше третий подход к разрешению конфликта управления, который называется **отложенным решением.**

Отложенный условный переход всегда выполняет следующую сразу за ним инструкцию, а условный переход осуществляется после задержки на эту одну инструкцию. Это все скрыто от программиста на языке ассемблера, потому что ассемблер может автоматически расположить инструкции в определенном порядке, чтобы получить от условного перехода то поведение, которое нужно программисту.

2.4.3 Динамическое прогнозирование условного перехода

Предположение о том, что переход не состоится, является одной из простейших форм прогнозирования условного перехода. В этом случае мы прогнозируем, что перехода не будет, сбрасывая конвейер, когда прогноз оказывается ошибочным. Для простого конвейера из пяти стадий такой подход, возможно, в совокупности с прогнозом компилятора, наверное, вполне адекватен. Для более глубоких конвейеров издержки условных переходов, измеренные в тактовых циклах, увеличиваются. Аналогично этому, при запуске сразу нескольких инструкций растут издержки перехода, измеряющиеся в потерянных инструкциях. Это сочетание означает, что в интенсивном конвейере простая статическая схема

прогнозирования будет, наверное, слишком сильно и неоправданно снижать производительность. Один из подходов для прогнозирования исхода условного перехода состоит в поиске адреса инструкции с целью определения, был ли осуществлен условный переход, когда инструкция выполнялась в последний раз, и если был, то приступать к извлечению. Такая технология называется **динамическим прогнозированием условного перехода.**

Одной из реализаций этого подхода является **буфер прогнозирования перехода**, или таблица истории переходов. Буфер прогнозирования перехода — это небольшой блок памяти, проиндексированный с использованием младшей части адреса инструкции условного перехода. Память содержит разряд, свидетельствующий о том, был недавно предпринят переход или нет.

Это простейший вид буфера, т.к. на самом деле мы не знаем, верен ли прогноз, он может быть помещен туда другим условным переходом, имеющим такие же младшие разряды адреса. Но это не влияет на точность. Прогнозирование — это всего лишь подсказка, по которой извлечение инструкции начинается в спрогнозированном направлении. Если окажется, что подсказка неверна, неправильно спрогнозированные инструкции удаляются, разряд прогнозирования инвертируется и сохраняется на прежнем месте, а затем извлекается и выполняется нужная последовательность инструкций.

Простая одноразрядная схема прогнозирования не дает заметного улучшения производительности: даже если переход происходит почти всегда, мы можем ошибиться в прогнозировании дважды, а не один раз, как в случае отсутствия перехода. Подобное затруднение демонстрируется в следующем примере.

Рассмотрим условный переход в цикле, который состоялся девять раз подряд, после чего не состоялся один раз. Какова точность прогнозирования для этого перехода, при условии, что разряд прогнозирования для него находится в буфере прогнозирования?

Ответ

Установившееся состояние прогнозирования приведет к неправильному прогнозированию первой и последней итерации цикла. Неверный прогноз последней итерации является неизбежным. Разряд прогнозирования будет показывать состоявшийся переход, поскольку переход на тот момент осуществлялся девять раз подряд. Неверный прогноз первой итерации дается из-за того, что разряд поменял свое значение при предыдущем выполнении последней итерации цикла, ведь при итерации с выходом из цикла переход не состоялся. Таким образом, точ-

ность прогнозирования того, что этот переход состоится, составляет в 90% случаев только 80% (два неверных прогноза при восьми верных).

В идеале для таких весьма регулярных условных переходов точность предсказателя соответствовала бы частоте этих переходов. Чтобы избавиться от этого недостатка, часто используются схемы двухразрядных предсказателей. В двухразрядной схеме предсказатель должен ошибиться дважды, прежде чем его значение изменится. На рис. 2.4.3 показан конечный автомат для двухразрядной схемы предсказателя.

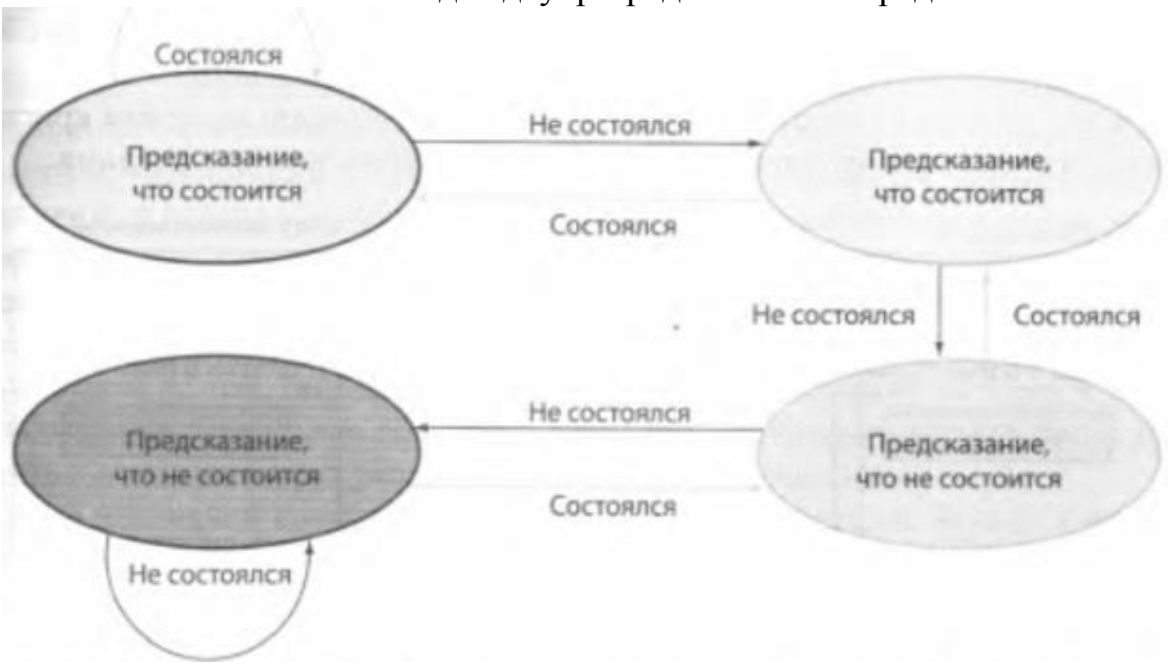


Рис. 2.4.3. Состояния в двухразрядной схеме прогнозирования

Буфер предсказателя условного перехода может быть реализован в виде небольшого специального буфера, доступного вместе с адресом инструкции на стадии конвейера IF. Если действие инструкции предсказывается состоявшимся, извлечение начинается с целевого адреса, как только он будет известей счетчику команд. Если предсказание окажется неверным, разряды предсказателя изменят свое значение, согласно рис. 2.4.3.

За счет использования двух разрядов вместо одного переход, который чаще всего является состоявшимся или несостоявшимся — будет неправильно спрогнозирован только один раз. Два разряда используются для кодирования четырех состояний системы, двухразрядная схема служит общим примером блока прогнозирования, основанного на работе счетчика, получающею приращение, когда прогноз точен, и получающего отрицательное приращение, когда он неточен, и использующего

среднее состояние своего диапазона чисел в качестве расхождении между прогнозированием состоявшегося и несостоявшегося перехода.

Компиляторы и ассемблеры пытаются поместить всегда выполняемые после перехода инструкции в слот задержки перехода. Задача программного обеспечения состоит в том, чтобы сделать последующие инструкции допустимыми и полезными. На рис. 2.4.4 показаны три способа диспетчеризации слота задержки перехода.

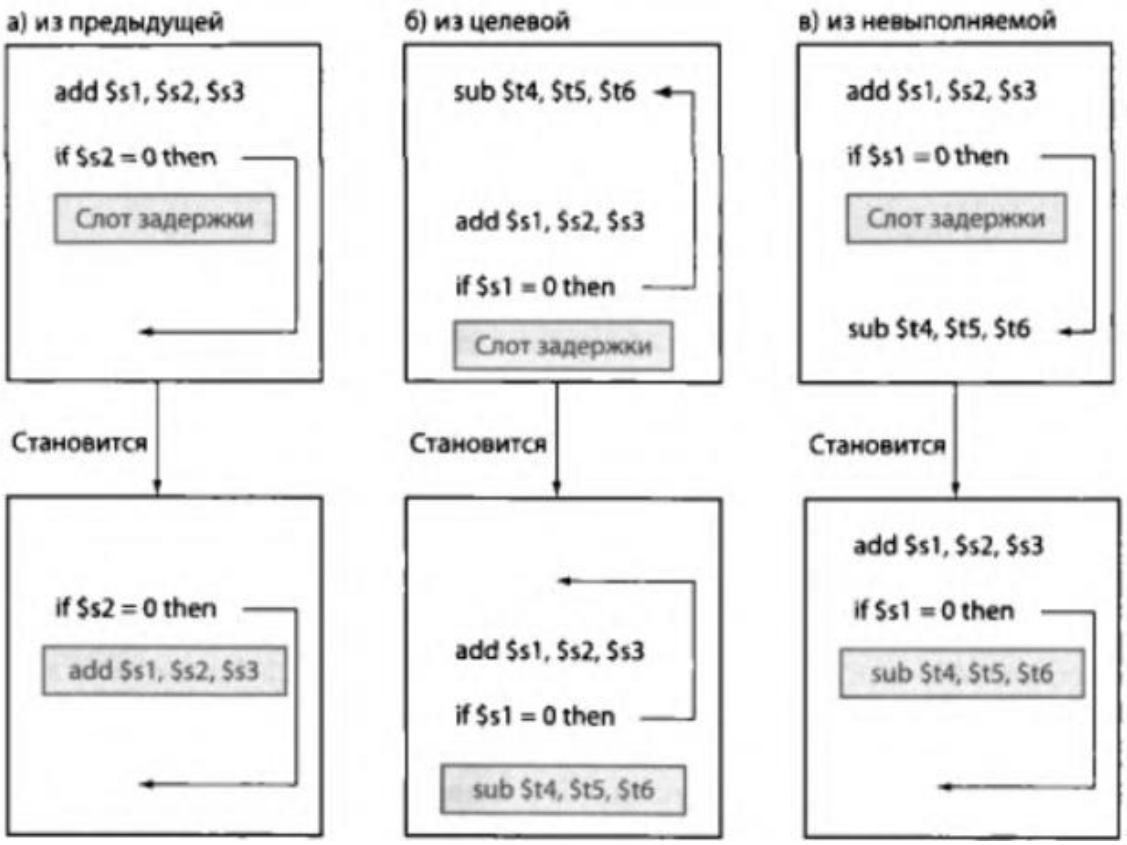


Рис. 2.4.4. Составление слота задержки перехода

Верхние блоки в каждой паре показывают код до составления, а нижние — после составления. В примере «а» слот задержки составлен из независимой инструкции, которая находилась до инструкции перехода. Это наилучший выход. Подходы «б» и «в» используются а том случае, когда подход «а» невозможен. В кодовой последовательности для примеров «б» и «в» использование $s1 в условии перехода не дает инструкции add (чьим регистром-получателем является Ss1) переместиться в слот задержки перехода. В кодовой последовательности для примера «б» слот задержки перехода составляется из инструкции, которая служит целью перехода; обычно целевая инструкция будет нуждаться в копировании, потому что по-другому добраться до нее невозможно,

Подход «б» является более предпочтительным, когда условный переход может состояться с высокой долей вероятности: например, если это переход при выполнении цикла. И наконец, слот задержки перехода может быть составлен из инструкции, не выполняемой при выходе из цикла, как в примере «в». Чтобы данная оптимизация было допустима для примера «б» или «в», должно быть вполне приемлемо выполнение инструкции sub, когда переход осуществляется в неожиданном направлении. Под приемлемостью мы понимаем, что работа будет проделана впустую, но на правильность выполнения программы это не повлияет, Так может получиться, к примеру, если на момент осуществления перехода в неожиданном направлении $t4 был неиспользованным временным регистром.

Ограничения, накладываемые диспетчеризацией отложенного перехода, возникают из-за: 1) ограничений, накладываемых на инструкции, вносимые в слоты задержки, и 2) возможности прогнозирования в процессе компиляции о том, будет ли, скорее всего, предпринят переход или нет.

Отложенный условный переход был простым и эффективным решением для конвейера, состоящего из пяти стадий и выполняющего по одной инструкции за каждый тактовый цикл. По мере того как процессоры переходят как на более длинные конвейеры, так и на выполнение сразу нескольких инструкций за один тактовый цикл, задержка условного перехода становится продолжительнее, а единственный слот задержки становится недостаточным. Следовательно, отложенный условный переход теряет популярность по сравнению с более дорогостоящими, но и более гибкими динамическими подходами.

Наряду с этим рост количества доступных транзисторов на одном кристалле привел к относительному удешевлению динамического прогнозирования.

Схема двухразрядного динамического прогнозирования использует только информацию о конкретном переходе. Исследователи заметили, что использование совместной информации о локальном переходе и о глобальном поведении недавно выполненных переходов приводит к более точным прогнозам при таком же количестве разрядов прогнозирования. Такие предсказатели называются сопоставляющими блоками прогнозирования. Типичный блок может иметь два двухразрядных предсказателя для каждого условного перехода, а выбор предсказателя может быть основан на том, был или не был предпринят последний условный переход. Таким образом, глобальное поведение может рас-

сматриваться как добавление дополнительных индексных разрядов для поиска предсказания.

Самой последней новинкой в прогнозировании переходов является использование соревновательных блоков прогнозирования. В них используется несколько предсказателей, и для каждого перехода отслеживается, какой из предсказателей выдал наилучшие результаты. Типичный соревновательный блок прогнозирования может содержать два предсказателя для каждого индекса условного перехода: один на основе локальной информации и один на основе глобального поведения условного перехода. Селектор выберет, какой предсказатель применить, для каждого отдельно взятого блока соревновательного прогнозирования. Аналогичным образом селектор может работать с одно- или двухразрядными предсказателями, выбирая тот из предсказателей, который был более точен. Подобные довольно сложные блоки прогнозирования используются в некоторых самых последних микропроцессорах.

2.4.4 Конфликты конвейера

При конвейеризации создаются ситуации, когда следующая инструкция не может выполняться в следующем тактовом цикле. Такие ситуации называются **конфликтами** и делятся на 3 типа:

Структурные конфликты – означает, что оборудование не может поддержать комбинацию инструкций, которую необходимо выполнять за один и тот же тактовый сигнал.

Конфликты данных – возникают в том случае, когда конвейер может быть остановлен из-за того, что один шаг должен будет ожидать завершение другого.

Конфликты управления – возникают из-за потребности принимать решение на основе результата выполнения одной инструкции во время выполнения другой инструкции.

2.5 Типы данных

Тип данных – характеристика набора данных, которая определяет:

- диапазон возможных значений данных из набора;
- допустимые операции, которые можно выполнять над этими значениями;
- способ хранения этих значений в памяти.

Типы данных можно разделить на две больших категории: **скалярные** и **векторные**.

Основными скалярными типами данных в компьютерах интеловской архитектуры являются: байт, слово, двойное слово и квадрослово (рис. 2.5.1).

Основными векторными типами данных являются: 64-х, 128-ми, 256-ти и 512-битные вектора, используемые в SIMD командах (рис. 2.5.1).

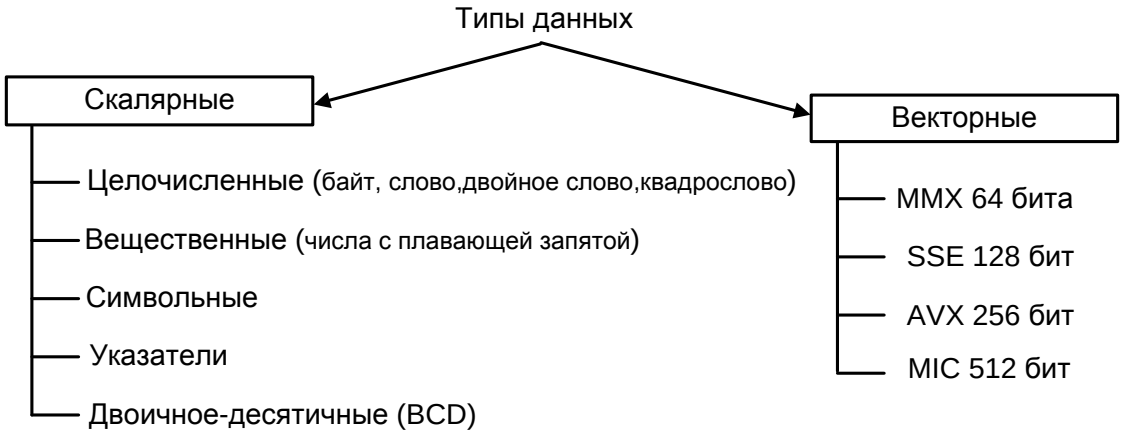


Рис. 2.5.1. Классификация типов данных

Целочисленные данные

Четыре формата данных (байт, слово, двойное слово, учетверенное слово) с фиксированной точкой (рис. 2.5.2) могут быть как со знаком, так и без знака.

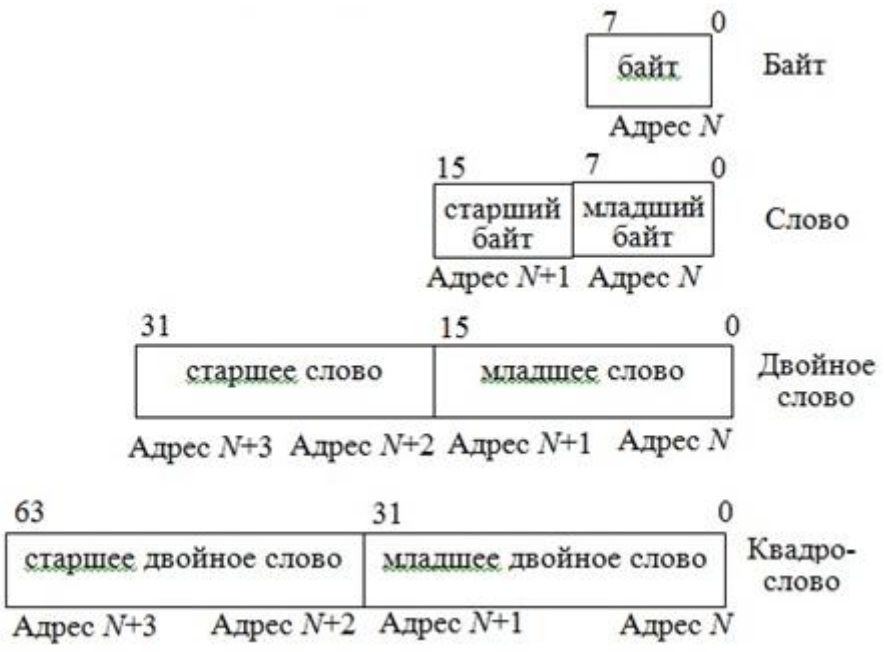


Рис. 2.5.2. Целочисленные данные

Под знак отводится старший бит формата данных. Представление таких данных и выполнение операций в арифметико-логическом устройстве (ALU) производится в дополнительном коде.

Данные в формате с плавающей точкой x87

Формат включает три поля: Знак (*S*), Порядок и Мантисса (рис. 2.5.3). Поле мантиссы содержит значащие биты числа, а поле порядка содержит степень 2 и определяет масштабирующий множитель для мантиссы. Форматы данных поддерживаются блоком обработки чисел с плавающей точкой (FPU).

Рис. 2.5.3. Форматы данных с плавающей точкой

Двоично-десятичные данные (BCD)

На рис. 2.5.4 приведены форматы двоично-десятичных данных.

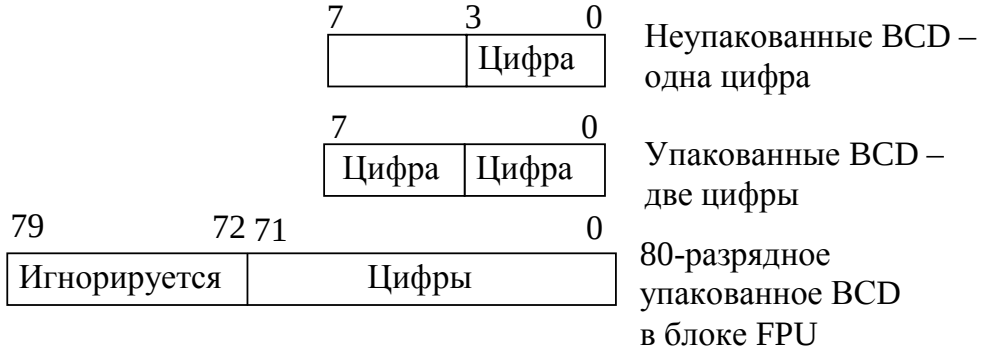


Рис. 2.5.4. Форматы двоично-десятичных данных

Данные типа строка

Строка представляет собой непрерывную последовательность бит, байт, слов или двойных слов (рис. 2.5.5). Строка бит может быть длиной до 1 Гбита, а длина остальных строк может составлять от 1 байта до 4 Гбайтов.

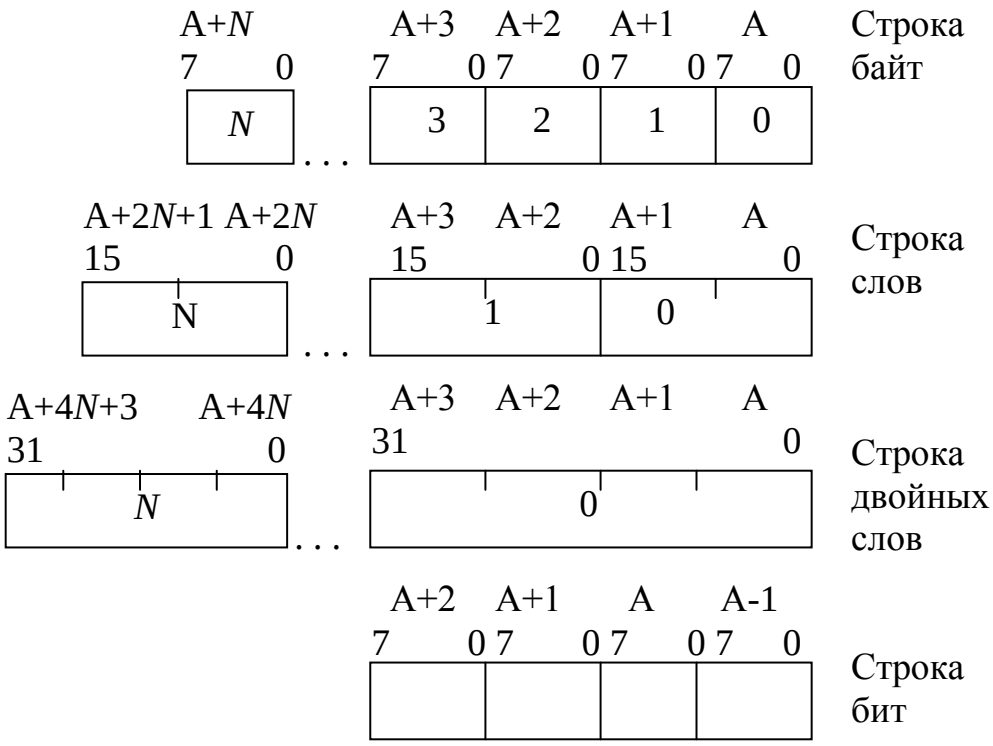


Рис. 2.5.5. Данные типа строка

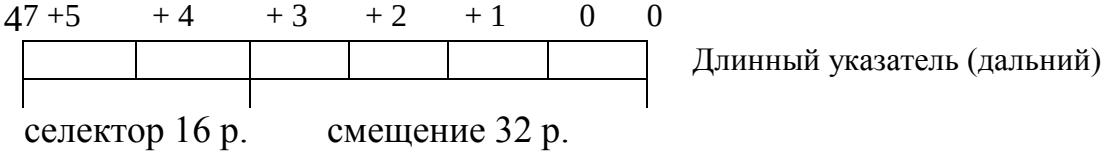
Символьные данные

Поддерживаются строки символов в коде ASCII и арифметические операции (сложение, умножение) над ними (рис. 2.5.6). Поддержка осуществляется блоком ALU.

Рис. 2.5.6. Символьные данные

Данные типа указатель

Указатель содержит величину, которая определяет адрес фрагмента данных. Поддерживается два типа указателей, приведенных на рис. 2.5.7.



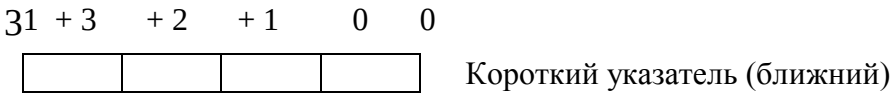


Рис. 2.5.7. Данные типа указатель

Векторные данные MMX-технологии

Целочисленные данные могут быть как со знаком, так и без знака (рис. 2.5.8).

Рис. 2.5.8. Данные MMX-технологии

Векторные данные SSE-расширения

На рис. 2.5.9 показаны 4 формата упакованных в 128 бит целочисленных данных, которые могут быть как со знаком, так и без знака.

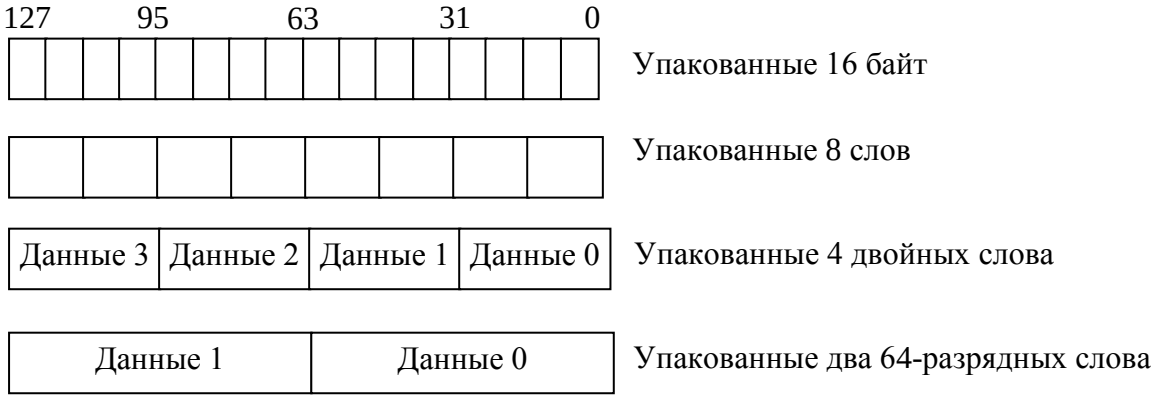


Рис. 2.5.9. Целочисленные данные SSE2 расширения

На рис. 2.5.10 приведен 128-разрядный формат упакованных данных с плавающей точкой одинарной и двойной точности.

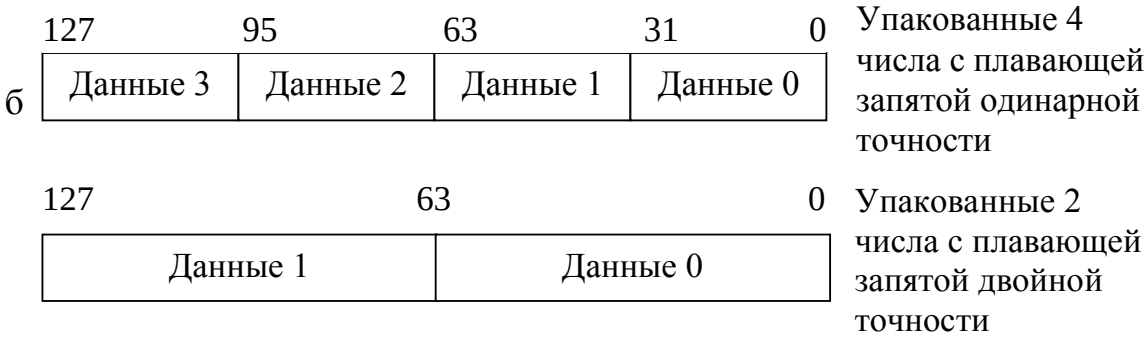


Рис. 2.5.10. Данные с плавающей запятой: а – для SSE-расширения; а, б – для SSE2-расширения

Векторные данные AVX-расширения

На рис. 2.5.11. показаны два формата упакованных в 256 бит чисел с плавающей запятой одинарной и двойной точности.

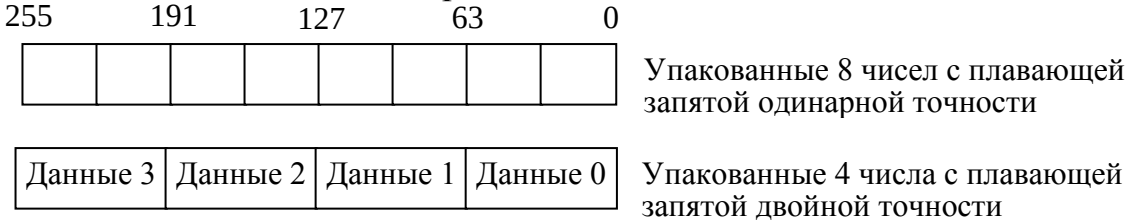


Рис. 2.5.11. Данные для AVX, AVX2-расширения

В AVX2-расширении кроме операций над числами с плавающей запятой, рассмотренных выше, выполняются логические операции OR, XOR, AND и т.д. над целочисленными данными (рис. 2.5.12). Арифметических операций нет.

Рис. 2.5.12. Данные AVX2-расширения

Векторные данные MIC-расширения

На рисуке 2.5.13. показаны два формата упакованных в 512 бит целочисленных данных и два формата чисел с плавающей запятой одинарной и двойной точности.

Рис. 2.5.13 Данные MIC-расширения

2.6 Структура и форматы команд ЭВМ

Все возможные преобразования дискретной информации могут быть сведены к четырем основным видам:

- передача информации в пространстве (из одного блока ЭВМ в другой);
- передача информации во времени (хранение);
- логические (поразрядные) операции;
- арифметические операции.

Величины, над которыми выполняются операции, могут быть скалярными (принимающими в каждый момент времени только одно значение) и векторными.

ЭВМ, являющаяся универсальным преобразователем дискретной информации, выполняет указанные виды преобразований.

Обработка информации (решение задач) в ЭВМ осуществляется автоматически путем программного управления. Программа представляет собой алгоритм обработки информации (решение задачи), записанный в виде последовательности команд, которые должны быть выполнены машиной для получения результата.

Команда представляет собой код, определяющий операцию и данные, участвующие в операции.

По характеру выполняемых операций различают следующие основные группы команд:

а) команды арифметических операций над числами с фиксированной и плавающей точками;

б) команды десятичной арифметики;

в) команды логических операций и сдвигов;

г) команды передачи кодов;

д) команды операций ввода/вывода;

е) команды передачи управления;

ж) команды векторной обработки;

з) команды задания режима работы машины и др.

Команда в общем случае состоит из операционной и адресной частей (рис. 2.6.1, *а*).

В свою очередь, эти части, что особенно характерно для адресной части, могут состоять из нескольких полей.

Операционная часть содержит код операции (КОП), который задает вид операции (сложение, умножение и др.). Адресная часть содержит информацию об адресах операндов и результате операции.

Структура команды определяется составом, назначением и расположением полей в команде.

Форматом команды называют ее структуру с разметкой номеров разрядов (бит), определяющих границы отдельных полей команды, или с указанием числа бит в определенных полях.

Важной и сложной проблемой при проектировании ЭВМ является выбор структуры и форматов команды, т.е. ее длины, назначения и размерности отдельных ее полей. Естественно стремление разместить в команде в возможно более полной форме информацию о предписываемой командой операции. Однако в условиях, когда в современных ЭВМ значительно возросло число выполняемых различных операций и соответственно команд (в системе команд x86 более 500 команд) и значительно увеличилась емкость адресуемой основной памяти (4 Гбайт, 6 Гбайт), это приводит к недопустимо большой длине формата команды.

Действительно, число двоичных разрядов, отводимых под код операции, должно быть таким, чтобы можно было представить все выполняемые машинные операции. Если ЭВМ выполняет M различных операций, то число разрядов в коде операции

$$n_{\text{коп}} \geq \log_2 M; \text{ например, при } M = 500\ n_{\text{коп}} = 9.$$

Если основная память содержит S адресуемых ячеек (байт), то для явного представления только одного адреса необходимо в команде иметь адресное поле для одного операнда с числом разрядов

$$n_{\text{A}} \geq \log_2 S; \text{ например, при } S = 4 \text{ Гбайт } \ n_{\text{A}} = 32.$$

Отмечавшиеся ранее, характерные для процесса развития ЭВМ расширение системы (наборы) команд и увеличение емкости основной памяти, а особенно создание микроЭВМ с коротким словом, потребова-

ли разработки методов сокращения длины команды. При решении этой проблемы существенно видоизменилась структура команды, получили развитие различные способы адресации информации.

Проследим изменения классических структур команд.

Чтобы команда содержала в явном виде всю необходимую информацию о задаваемой операции, она должна, как это показано на рис. 2.6.1, *б*, содержать следующую информацию:

A_1, A_2 – адреса операндов, A_3 – адрес результата, A_4 – адрес следующей команды (принудительная адресация команд).

Такая структура приводит к большой длине команды (например, при $M = 500$, $S = 4$ Гб длина команды – 137 бит) и неприемлема для прямой адресации операндов основной памяти. В компьютерах с RISC-архитектурой четырехадресные команды используются для адресации операндов, хранящихся в регистровой памяти процессора.

Можно установить, что после выполнения данной команды, расположенной по адресу K (и занимающей L ячеек), выполняется команда из $(K + L)$-й ячейки. Такой порядок выборки команды называется естественным. Он нарушается только специальными командами передачи управления. В таком случае отпадает необходимость указывать в команде в явном виде адрес следующей команды.

В трехадресной команде (рис. 2.6.1, *в*) первый и второй адреса указывают ячейки памяти, в которых расположены операнды, а третий определяет ячейку, в которую помещается результат операции.

Можно условиться, что результат операции всегда помещается на место одного из операндов, например первого. Получим двухадресную команду (рис. 2.6.1, *г*), т.е. для результата используется подразумеваемый адрес.

В одноадресной команде (рис. 2.6.1, *д*) подразумеваемые адреса имеют уже и результат операции, и один из операндов. Один из операндов указывается адресом в команде, в качестве второго используется содержимое регистра процессора, называемого в этом случае регистром результата, или аккумулятором. Результат операции записывается в тот же регистр.

Наконец, в некоторых случаях возможно использование безадресных команд (рис. 2.6.1, *е*), когда подразумеваются адреса обоих операндов и результата операции, например при работе со стековой памятью.

С точки зрения программиста, наиболее естественны и удобны трехадресные команды. Однако из-за необходимости иметь большее число разрядов для представления адресов основной памяти и кода операции длина трехадресной команды становится недопустимо большой и

ее не удается разместить в машинном слове. Следует отметить, что очень часто в качестве операндов используются результаты предыдущих операций, хранимые в регистрах машины. По указанным причинам в современных ЭВМ применяют трехадресные команды для адресации регистров. Обычно в ЭВМ используется несколько структур и форматов команд.

Приведенные на рис. 2.6.1 структуры команд достаточно схематичны. В действительности адресные поля команд большей частью содержат не сами адреса, а только информацию, позволяющую определить действительные (исполнительные) адреса операндов в соответствии с используемыми в командах способами адресации (подробнее в разделе 2.7.).

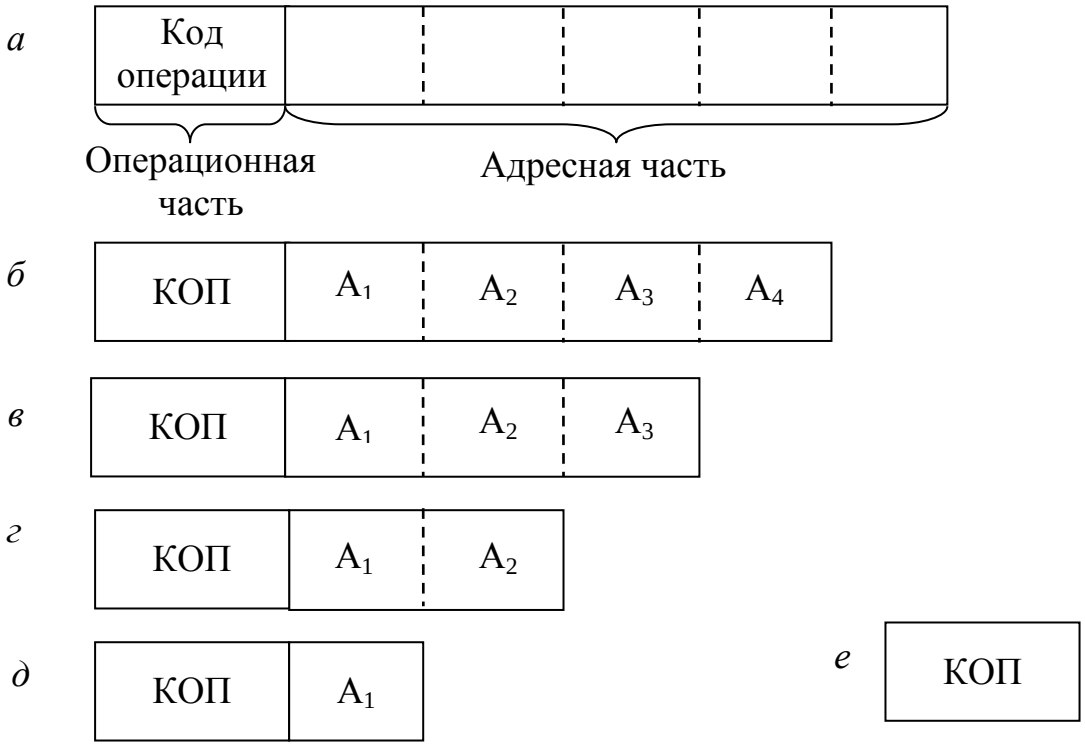


Рис. 2.6.1. Структуры команд:
а – обобщенная; *б* – четырехадресная; *в* – трехадресная;
г – двухадресная; *д* – одноадресная; *е* – безадресная

2.6.1 Обобщенный формат команд x86

Базовый набор команд 32-разрядного интеловского процессора обеспечивает выполнение операций над операндами, которые находятся в регистре, памяти или непосредственно в команде. В набор входят безадресные, одно-, двух- и трехадресные команды. Процессор реализует

следующие шесть типов двухадресных команд: регистр – регистр; память – регистр; непосредственный операнд – регистр; регистр – память; память – память; непосредственный операнд – память.

Операнды могут содержать 8, 16 или 32 разряда. Для реализации различных типов команд определены форматы, задающие порядок размещения информации о выполняемой операции и способах выбора операндов.

Обобщенный вид формата команды показан на рис. 2.20. Он допускает наличие следующих полей: кода операции (1 или 2 байта); байтов адресации (0, 1 или 2 байта); байтов смещения (0, 1, 2 или 4 байта); байтов непосредственных данных – операндов (0, 1, 2 или 4 байта).

Команды содержат от 1 до 12 байт. Проведенные оценки показывают, что в среднем длина команды составляет 4–5 байт.

Рассмотрим назначение основных полей кода команды (рис. 2.6.2). Код операции (КОП) определяет тип выполняемой операции, а также в некоторых командах в первом байте может содержаться бит W, задающий разрядность операндов:

W = 0 – операция с байтами;

W = 1 – операция со словами (16 или 32 разряда).

КОП	Байты адресации		Смещение	Операнд
	MOD R/M	SIB		
1 или 2 байта	0 или 1 байт	0 или 1 байт	0, 1, 2 или 4 байта	0, 1, 2 или 4 байта

Рис. 2.6.2 Общий формат команд

В ряде команд первый байт КОП содержит поля reg или sreg, определяющие адреса используемых регистров. Трехбитовое поле reg задает выбираемый регистр в соответствии с разрядностью обрабатываемых операндов. Поле sreg (двух или трехбитовое) определяет адрес сегментных регистров.

Байт адресации MOD R/M содержит три поля (рис. 2.6.3). Поля MOD и R/M задают адрес одного из операндов, который может храниться в регистре или ячейке памяти. Кодировка этих полей определяет выбираемый способ адресации.

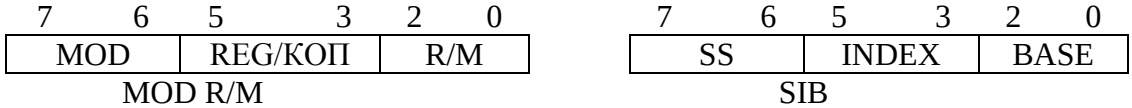


Рис. 2.6.3. Форматы байтов MOD R/M и SIB

В одноадресных командах поле REG/КОП содержит дополнительные биты кода операции. В двухадресных командах поле REG содержит адрес регистра, в котором хранится второй из операндов. Тип команды (одно- или двухадресная) определяется первым битом КОП. Поле MOD указывает, какой разрядности смещение используется для формирования адреса. Если оно имеет значение 00 (при некоторых значениях R/M) или 01, 10, то используется 8-, 16- или 32-разрядное смещение. Это смещение задается соответствующими байтами в коде команды, которые располагаются после байтов адресации.

Для реализации некоторых способов относительной адресации используется байт SIB. Он содержит 3-битовые поля INDEX и BASE, определяющие выбор регистров, используемых в качестве индексного и базового регистров, и поле SS, задающее масштабный коэффициент для модификации значения индекса.

При выполнении операций с непосредственной адресацией один из операндов задается в последних байтах команды (рис. 2.6.2). В этом случае КОП ряда команд содержит бит *S*, определяющий способ использования непосредственно задаваемых данных.

2.6.2 Развитие системы команды x86

В качестве примера рассмотрим набор команд и способы адресации, используемые в процессорах интеловской архитектуры. Для этих процессоров в табл. 2.6.1 приведены данные о развитии их системы команд x86 процессоров Intel.

Таюлица .2.6.1. Развитие системы команд x86 процессоров

Год появления набора команд	Тип процессора, где набор был реализован впервые	Общее число команд	Смысл расширения
1979	i8086	170	Исходный набор команд x86
1985	i386	220	50 новых команд для перехода к архитектуре IA-32
1997	Pentium/MMX	277	57 MMX-команд
1999	Pentium III	347	70 команд SSE-расширения, AMD 3DNow!
2000	Pentium 4 Northwood	491	144 команды SSE2
2004	Pentium 4 Prescott	504	13 команд SSE3, AMD x86-64
		514	10 команд Intel VT-x, AMD-V, Intel EM64T
2006	Core2 Duo (65 нм)	546	32 команды SSSE3
2007	Penryn (45 нм)	593	47 команд SSE4.1

Год появления набора команд	Тип процессора, где набор был реализован впервые	Общее число команд	Смысл расширения
2009	Core i7 Nehalem (45нм)	600	7 команд SSE4.2
2010	Core i5 Westmere (32 нм)	606	6 команд AES-NI
2011	Core i7, i5 Sandy Bridge (32 нм)		AVX
2013	Core i7, i5 Haswell (22 нм)		AVX2, FMA3, BMI
2015	Core i7, i5 Skylake (14 нм)		MIC, SGX

Изначально в базовом наборе команд процессора i8086 были предусмотрены команды обработки чисел с плавающей запятой, которые до i386-процессора включительно выполнялись на дополнительном сопроцессоре. Начиная с процессора i486, блок обработки чисел с плавающей запятой (FPU) стал составной частью микропроцессора.

Переход на 32-разрядную интеловскую архитектуру (IA-32) был осуществлен в процессоре i386 с добавлением 50 новых команд. Все последующие модели процессоров (до Pentium 4 включительно) имеют IA-32 архитектуру, несмотря на то, что расширение системы команд происходило неоднократно.

Технология виртуализации Intel VT-x, разработанная для IA-32 и поддерживаемая современными процессорами, использует 10 новых инструкций VM.

В 2002 г. впервые со времен i386 архитектура x86 подвергается принципиальным изменениям. Разработчиками фирмы AMD была создана 64-разрядная архитектура, получившая название «x86-64». Эта архитектура базируется на существующей архитектуре IA-32. В 2004 г. Intel вводит в серверные процессоры Xeon 64-разрядную технологию EM64T (Extended Memory), с программной точки зрения практически идентичную той, что предложила AMD. С 2006 г. эта технология под названием «Архитектура Intel 64» начала использоваться в клиентских ПК с процессором Core 2 Duo.

Начиная с 1997 г. и по сей день система команд x86 расширяется за счет SIMD инструкций: **MMX, SSE, SSE2, SSE3, SSSE3, SSE4, AVX, AVX2, MIC**.

В 2010 г. в процессорах Intel семейства Westmere (32 нм) стали использоваться 6 новых инструкций SIMD, которые Intel назвал AES-NI.

В основе технологии MMX лежит расширение набора команд (57 новых команд) для эффективного выполнения типичных мультимедийных алгоритмов, к числу которых относятся и многие алгоритмы, характерные для цифровой обработки сигналов. Это первое существен-

ное изменение в системе команд микропроцессоров семейства x86, начиная с выхода в свет микропроцессора i386. В технологии MMX использована модель обработки данных SIMD, предусматривающая одновременное выполнение операции над несколькими целочисленными операндами разрядностью 1, 2 или 4 байта.

MMX-команды используют восемь 64-разрядных MMX-регистров с плавающей запятой и реализуются в том же режиме процессора, что и команды с плавающей запятой. MMX-команды делятся на следующие группы: арифметические, логические и сдвига, сравнения, передачи данных, упаковки и распаковки, отмены режима MMX. Все программное обеспечение, созданное для ранее выпущенных процессоров, без всяких изменений может выполняться на процессорах с технологией MMX.

Стремясь устранить недостатки, свойственные MMX-технологии (отсутствие MMX-команд для работы с плавающей запятой, невозможность выполнения операций с плавающей запятой при выполнении MMX-команд), Intel решила внести необходимые дополнения в архитектуру процессора Pentium III.

SSE-расширение

Новые 70 команд SSE-расширения делятся на 4 категории:

- SIMD-команды обработки данных в формате с плавающей запятой одинарной точности (SPFP-команды);
- дополнительные SIMD-команды для обработки целочисленных данных;
- команды управления кэшированием;
- команды сохранения и восстановления состояния процессора.

SPFP-команды используют 8 новых 128-разрядных регистров (XMM-регистры) и новый тип данных – 128-разрядное значение, содержащее 4 последовательно расположенных («упакованных») 32-разрядных числа с плавающей запятой одинарной точности. При выполнении инструкций с XMM традиционное оборудование FPU/MMX не используется, что позволяет эффективно смешивать инструкции MMX с инструкциями над операндами с плавающей точкой.

Большинство SPFP-команд имеют два операнда. Данные, содержащиеся в первом операнде, после выполнения команды, как правило, замещаются результатами, а данные, содержащиеся во втором операнде, остаются неизменными.

SPFP-команды поддерживают два типа операций над упакованными данными с плавающей запятой – параллельные и скалярные. Парал-

лельные операции выполняются над четырьмя 32-разрядными элементами данных, упакованными в каждый 128-разрядный операнд.

Скалярные операции выполняются над младшими (занимающие разряды 0–31) элементами данных двух операндов. Остальные три элемента данных не изменяются.

В расширение SSE включены дополнительные SIMD-команды для работы с целочисленными данными. Эти новые команды расширяют возможности существующего набора команд технологии MMX. Они выполняют SIMD-операции над несколькими целочисленными данными, упакованными в 64-разрядные группы, загружают и хранят упакованные данные в 64-разрядных MMX-регистрах.

Кроме того, в SSE введены команды нового типа, обеспечивающие:

- управление кэшированием данных с целью повышения эффективности использования кэш-памяти и сокращения числа обращений к основной памяти;
- упреждающее кэширование данных с целью организации параллельной работы конвейера команд и обмена с памятью.

Первая группа команд выполняет запись данных из MMX (XMM) регистра в память, минуя кэш. Вторая – обеспечивает запись данных из памяти в кэш различных уровней.

Кроме XMM-регистров в микропроцессоре Pentium III появился новый регистр состояния и управления MXCSR. Для работы с этими регистрами требуется поддержка как со стороны процессора, так и со стороны операционной системы. Чтобы прикладные программы и ОС могли сохранять и восстанавливать состояния новых компонентов процессора, введено несколько команд управления. Первая группа команд управления обеспечивает сохранение в памяти содержимого регистра MXCSR и, наоборот, загружает слово состояния из памяти в регистр MXCSR. Вторая группа – сохраняет в памяти состояние процессора (состояние регистров данных FPU, MMX-регистров, XMM-регистров) и восстанавливает ранее сохраненное состояние процессора.

Расширения SSE2, SSE3, SSSE3, SSE4

Расширение SSE2, введенное в состав Pentium 4 Northwood, значительно расширяет возможности обработки нескольких операндов по принципу SIMD по сравнению с SSE. В нем используется 144 новых команды, обеспечивающих одновременное выполнение операций над несколькими операндами, которые располагаются в памяти и в 128-разрядных регистрах XMM. В регистрах могут храниться и одновременно обрабатываться два числа с плавающей запятой в формате двой-

ной точности (64 разряда) или 4 числа в формате одинарной точности (32 разряда), любые целочисленные типы данных, способные разместиться в 128-разрядных регистрах.

Расширение SSE2, представляя собой симбиоз MMX и SSE, обладает большей гибкостью и позволяет добиваться впечатляющего прироста производительности. Команды SSE2 существенно повышают эффективность процессора при реализации трехмерной графики и Интернет-приложений, обеспечение сжатия и кодирования аудио- и видеоданных и в ряде других приложений.

Расширение SSE3, введенное в состав Pentium 4 Prescott, включает 5 новых операций с комплексными числами, 5 потоковых операций над числами с плавающей запятой, 2 команды для синхронизации потоков и одну специальную инструкцию для применения при кодировании видео.

Расширение SSSE3 (Supplemental SSE3 – дополнительное потоковое SIMD-расширение 3) появилось в процессорах с микроархитектурой Intel Core и поддерживается процессором Intel Atom. Новыми в SSSE3, по сравнению с SSE3, являются 16 уникальных команд, работающих с упакованными целыми данными. Каждый из них может работать как с 64-битными (MMX), так и с 128-битными (XMM) регистрами, поэтому Intel в своих материалах ссылается на 32 новые команды. Новые инструкции включают работу со знаком, сдвиги, перемешивание байт, умножение, горизонтальное сложение/вычитание целых.

Расширение SSE4.1 появилось в первом процессоре Intel с 45 нм техпроцессом (кодовое наименование Penryn). Набор команд SSE4.1 включает 47 новых инновационных инструкций, основными из которых являются примитивы векторизации для компиляторов и ускорители кодирования видеозаписей с высоким расширением и обработки фотоизображений.

Расширение SSE4.2 разработано Intel для процессоров с новой микроархитектурой Nehalem. Введенные в набор SSE4.2 инструкции ориентированы на ускорение обработки строк и текстовой информации.

Ни одна из SSE4 инструкций не работает с 64-битными MMX-регистрами, только с 128-битными XMM-регистрами.

Расширения AES-NI

Расширение AES-NI (Advanced Encryption Standard New Instructions) – набор из 6 новых SIMD-инструкций, ускоряющий процесс шифрования и дешифрования информации по стандарту AES. Стандарт AES является стандартом шифрования США, принятым в 2000 г. Он специфицирует алгоритм Rijndael, который представляет собой симметричный блочный шифр, работающий с блоками длиной 128 бит, и ис-

пользует ключи длиной 128, 192 и 256 бит. По заявлению правительства США для взлома шифрования при использовании 128-битного ключа потребуется 149 триллионов лет.

Расширения AVX

Новое расширение AVX (Advanced Vector Extensions) – расширение системы команд x86 для микропроцессоров с новой микроархитектурой Intel Sandy Bridge (2010 г.) и процессоров AMD Bulldozer (2011 г.), анонсированная Intel в 2008 г., представляет различные улучшения, новые инструкции и новую схему кодирования машинных кодов. Размер векторных регистров SIMD увеличивается со 128 (XMM) до 256 бит (регистры YMM). Существующие 128-битные инструкции будут использовать только младшую половину новых YMM-регистров. В будущем возможно расширение до 512 или 1024 бит. Набор инструкций AVX позволяет использовать любую двухоперандную инструкцию XMM в трехоперандном виде без модификации двух регистров-источников, с отдельным регистром для результата. Добавлены инструкции с количеством операндов более трех. Новая система кодирования машинных кодов VEX предоставляет новый набор префиксов кода, которые расширяют пространство возможных машинных кодов. Использование YMM-регистров поддерживают операционные системы: Windows 7, Windows Server 2008 R2, Linux (версия ядра 2.6.30).

Расширение AVX подходит для интенсивных вычислений с плавающей точкой в мультимедийных, научных и финансовых задачах. Увеличивает степень параллелизма и пропускную способность в вещественных SIMD-вычислениях.

Расширения AVX2, FMA3, BMI

Набор инструкций AVX2 является расширением набора AVX и используется в процессорах с новой микроархитектурой Haswell. Главное отличие нового набора инструкций AVX2 от прежней версии AVX заключается в том, что если ранее 256-битные операции с AVX регистрами были доступны только для операнда с плавающей запятой, а для целочисленных операндов были доступны лишь 128-битные операции, то в AVX2 256 операции стали доступны и для целочисленных операндов.

Кроме того в AVX2 появилась улучшенная поддержка сдвигов и перестановок в векторных операциях. Есть и новые инструкции, используемые для сборки нескольких (4-х или 8-ми) несвязанных элемен-

тов в один векторный элемент, благодаря чему есть возможность более полно загружать 256-битные AVX регистры.

Новый **набор инструкций FMA3 (Fused Multiply Add)** используется в процессорах с новой микроархитектурой Haswell и предназначен для проведения операций совмещённого умножения и сложения над 3-мя операндами. Использование операций FMA3 позволяет более эффективно реализовать операции деления, извлечение квадратного корня, умножения векторов и матриц и т.д. Набор FMA3 включает 36 инструкций с плавающей запятой для выполнения 256-битных вычислений и 60 инструкций для 128-битных векторов.

В **набор команд BMI** (Bit Manipulation Instructions) входят 15 скалярных инструкций для битовых операций, которые работают с целочисленными регистрами общего назначения.

Эти инструкции разбиты на три группы:

- манипуляции на отдельными битами, такие как вставка, сдвиг и извлечение бит;
- подсчет битов, например, подсчет ведущих нулей в записи чисел;
- целочисленное умножение произвольной точности.

Данный набор инструкций позволяет ускорять ряд специфических операций, используемых, например, при шифровании.

Расширения MIC, SGX

В 2015 г. прошла презентация новой микроархитектуры процессоров Intel Skylake с техпроцессом 14 нм. В презентации прозвучали достаточно любопытные откровения о том, что построенные на Intel Skylake серверные и клиентские процессоры могут серьезно различаться по своей конфигурации даже на уровне микроархитектуры. Один пример такого отличия уже хорошо известен – серверные Skylake получили поддержку расширения системы команд MIC (AVX-512), которое в остальных процессорах не используется.

Расширение SIMD инструкций MIC (Many Integrated Core Architecture) использует шестнадцать 512-битных регистров для выполнения векторных операций над целочисленными данными и данными с плавающей запятой одинарной и двойной точности.

Нововведения в системе команд не миновали и клиентские процессоры. Так, в них появились новые инструкции семейства Intel SGX (Software Guard Extension). Входящие в этот набор команды позволяют приложению создать для своего исполнения изолированную и защищённую среду в памяти, доступ к которой будет невозможен ни для каких иных процессов и устройств. Таким образом, приложение, опери-

рующее критически важной информацией, сможет защитить свой код и данные от каких-либо программных и аппаратных атак и вторжений, что может поднять безопасность платформы x86 на новый уровень. Intel отдельно подчеркивает, что благодаря SGX можно создавать и полностью защищенный программный код, который невозможно отслеживать с помощью аппаратных отладчиков ITP-класса.

2.7 Способы адресации команд и данных

Интеловский 32-разрядный процессор реализует сегментную организацию оперативной памяти, при которой физический адрес ячейки памяти формируется путем сложения базового адреса сегмента и относительного адреса ячейки внутри сегмента.

Базовый адрес определяется содержимым 16-разрядного сегментного регистра и зависит от режима работы процессора. Если он работает в режиме обработки 16-разрядных данных (режим реальных адресов), то 20-разрядный базовый адрес формируется путем сдвига содержимого сегментного регистра на 4 разряда влево. Если процессор работает в режиме обработки 32-разрядных данных (защищенный режим), то 32-разрядный базовый адрес содержится в дескрипторе, выбор которого из таблицы дескрипторов осуществляется с помощью селектора – содержимого соответствующего сегментного регистра.

В качестве относительного адреса используется содержимое регистров общего назначения или эффективный адрес (EA), который формируется в соответствии с заданным способом адресации. EA является 16- или 32-разрядным и формируется в зависимости от значения полей MOD и R/M и содержимого байта SIB (для 32-разрядных адресов). В общем случае EA образуется путем арифметического сложения трех компонентов:

- содержимого базового регистра;
- содержимого индексного регистра;
- 8-, 16-, 32-разрядного смещения, заданного в одном, двух или четырех байтах команды.

В зависимости от значений полей MOD и R/M для формирования EA используются все или часть этих слагаемых.

В процессоре осуществляются следующие способы адресации операндов:

- непосредственная адресация;
- регистровая адресация;
- косвенно-регистровая адресация;
- прямая адресация;

- базовая адресация;
- индексная адресация;
- базово-индексная адресация;
- базово-индексная адресация со смещением.

В современных микропроцессорах Intel Core i5, i7 базовый набор команд и используемые способы адресации операндов практически полностью совпадают с набором команд и способов адресации в предыдущих моделях – Core 2 Duo, Pentium 4. Процессоры обеспечивают реальный и защищенный режимы работы, реализуют сегментную и страничную организации памяти. Таким образом, пользователь имеет дело с хорошо знакомым набором регистров и способов адресации, может работать с базовой системой команд и известными вариантами реализации прерываний и исключений, которые характерны для всех моделей семейств Intel Core и Pentium.

Существует два различных принципа поиска операндов в памяти: **ассоциативный** и **адресный.**

Ассоциативный поиск операнда (рис. 2.7.1) предполагает одновременный просмотр содержимого всех ячеек памяти для выявления кода, содержащего заданной командой ассоциативный признак (**тег**). Этот код и выбирается из памяти в качестве искомого операнда. В современных компьютерах ассоциативная выборка используется в кэш-памяти.

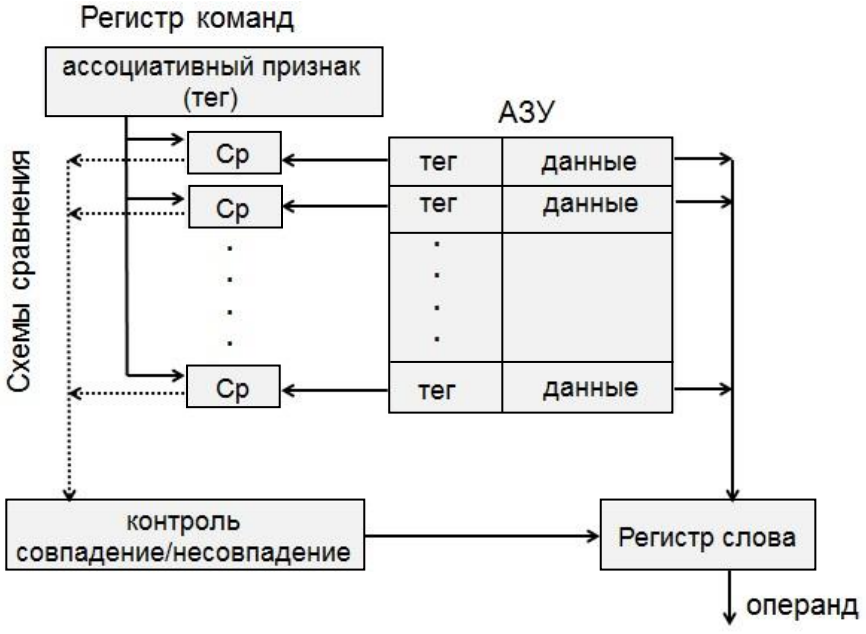


Рис. 2.7.1. Ассоциативный поиск

Адресный поиск предполагает, что искомый операнд извлекается из ячейки, номер которой формируется на основе информации в адресном поле команды (рис.2.7.2).

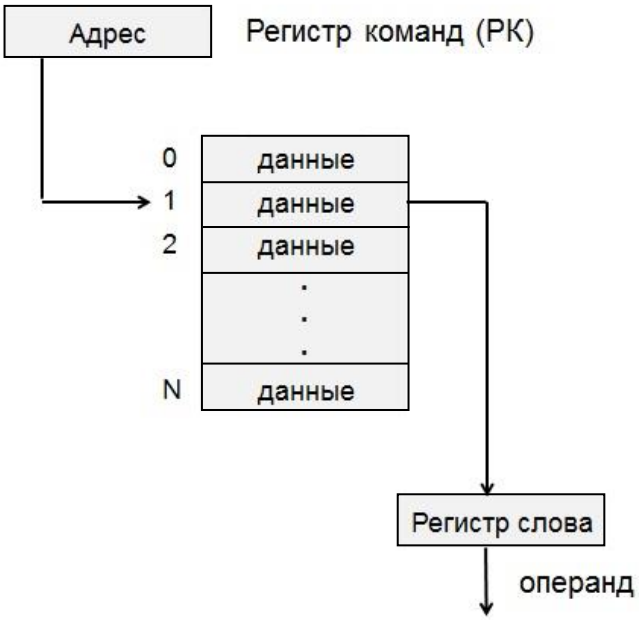


Рис. 2.7.2 Адресный поиск

Ниже мы будем рассматривать только реализацию адресного принципа поиска операнда. Следует различать понятия «адресный код» в команде $А_К$ и «исполнительный (физический) адрес» $А_И$. **Адресный код** – это информация об адресе операнда, содержащаяся в команде. **Исполнительный адрес** – это номер ячейки памяти, к которой производится фактическое обращение. В современных ЭВМ адресный код, как правило, не совпадает с исполнительным адресом. Таким образом, способ адресации можно определить как способ формирования исполнительного адреса операнда $А_И$ по адресному коду команды $А_К$.

Способов адресации существует много. Параметры процесса обработки информации существенно зависят от выбранного способа адресации. Одни способы адресации позволяют увеличить объём адресуемой памяти без удлинения команды, но снижают скорость выполнения операции, другие ускоряют операции над массивами данных, третьи – упрощают работу с подпрограммами и т.д.

В системах команд современных ЭВМ часто предусматривается возможность использования нескольких способов адресации операндов для одной и той же операции. Для указания способа адресации вводятся дополнительные разряды в поле кода операции, длина которого при этом возрастает.

Адресация операнда в команде может быть **явной** или **неявной.** При явной адресации в команде есть поле адреса операнда, в котором задается адресный код $A_К$. Большинство методов адресации являются явными.

При неявной адресации адресное поле в команде отсутствует, адрес операнда подразумевается кодом операции. Метод неявной адресации операндов используется во всех процессорах. Основное его назначение – уменьшение длины команды за счет исключения части адресов. При этом методе код операции точно задает адрес операнда. Например, из команды исключается адрес приемника результата. При этом подразумевается, что результат в этой команде помещается на место второго операнда.

Способы формирования адресов ячеек памяти ($A_И$) можно разделить на абсолютные и относительные.

Абсолютные способы формирования исполнительного адреса

Абсолютные способы формирования предполагают, что двоичный код адреса ячейки памяти ($A_И$) может быть извлечен целиком из адресного поля команды или из какой-либо другой ячейки (регистра), никаких преобразований над кодом адреса не производится.

К абсолютным способам относятся непосредственная, прямая и косвенная адресации, которые имеют различную кратность обращения (R) к памяти.

Непосредственная адресация операнда

При этом способе операнд располагается в адресном поле команды. Обращение к регистровой памяти (РП) или ОП за операндом не производится ($R = 0$), он выбирается вместе с командой. Таким образом, уменьшается время выполнения операции, сокращается объем памяти. Непосредственная адресация удобна для задания констант, длина которых меньше или равна длине адресного поля команды.

Прямая адресация операндов

При этом способе (рис. 2.7.3) адресации обращение за операндом в РП или ОП производится по адресному коду в поле команды ($R = 1$), т.е. исполнительный адрес операнда совпадает с адресным кодом команды ($A_И = A_К$).

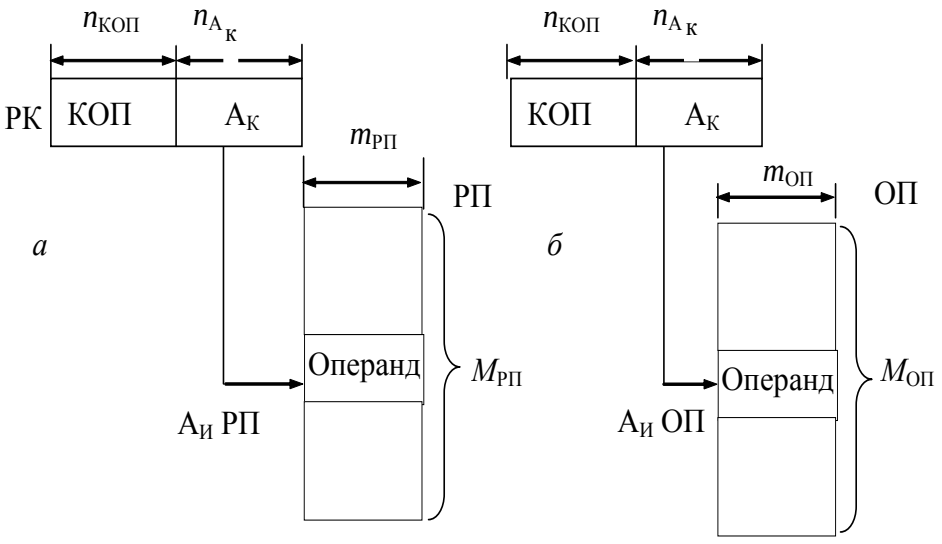


Рис. 2.7.3. Схема прямой адресации:
а – к регистровой памяти; *б* – к основной памяти

Обеспечивая простоту программирования, этот метод имеет существенный недостаток. Для адресации к ячейкам памяти большой емкости требуется «длинное» адресное поле в команде. Прямая адресация широко используется в сочетании с другими способами адресации. В частности, вся адресация к «малой» регистровой памяти ведется только с помощью прямой адресации.

Косвенная адресация операндов

При этом способе адресный код команды указывает адрес ячейки (регистра) памяти, в которой находится не сам операнд, а лишь адрес операнда, называемый указателем операнда. Адресация к операнду через цепочку указателей (косвенных адресов) называется косвенной ($R \geq 2$).

Адрес указателя, задаваемый программой, остается неизменным, а косвенный адрес может изменяться в процессе выполнения программы. Косвенная адресация, таким образом, обеспечивает переадресацию данных, т.е. упрощает обработку массивов и списковых структур данных, упрощает передачу параметров подпрограммам, но не обеспечивает перемещаемость программ в памяти.

Косвенная адресация также широко используется в ЭВМ, имеющих короткое машинное слово, для преодоления ограничений короткого формата. В этом случае первый указатель должен располагаться в РП (рис. 2.7.4).

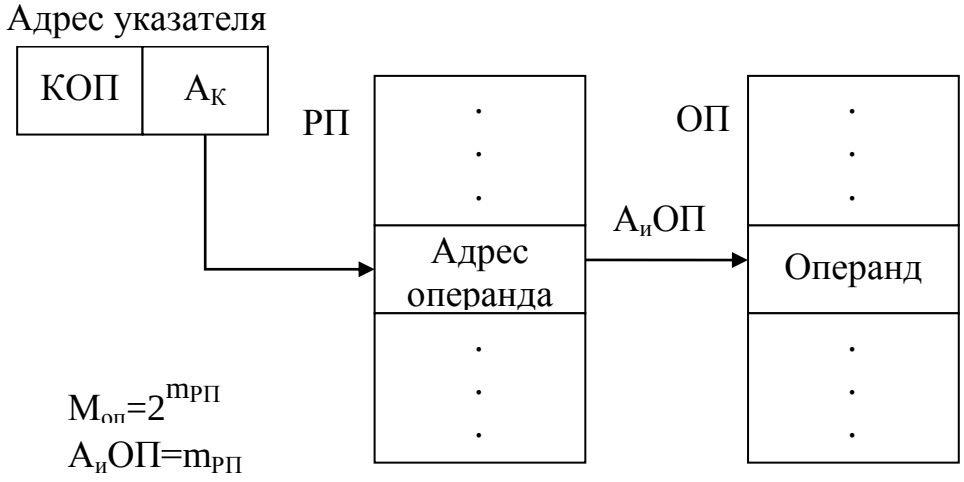


Рис. 2.7.4. Схема косвенной адресации

Относительные способы формирования исполнительных адресов ячеек памяти

Относительные способы формирования $A_И$ предполагают, что двоичный код адреса ячейки памяти образуется из нескольких составляющих: Б – код базы, И – код индекса, С – код смещения, используемых в сочетаниях (Б и С), (И и С), (Б, И и С).

При относительной адресации применяются два способа вычисления адреса $A_И$:

- суммирование кодов составляющих адреса;
- совмещение (конкатенация) кодов составляющих адреса.

Суммирование кодов составляющих производится для случаев:

$$A_И = Б + С;\ A_И = И + С;\qquad A_И = Б + И + С.$$

Базирование способом суммирования

В команде адресный код $A_К$ разделяется на две составляющие: $A_Б$ – адрес регистра регистровой памяти, в котором хранится база Б (базовый адрес); С – код смещения относительно базового адреса (рис. 2.7.5).

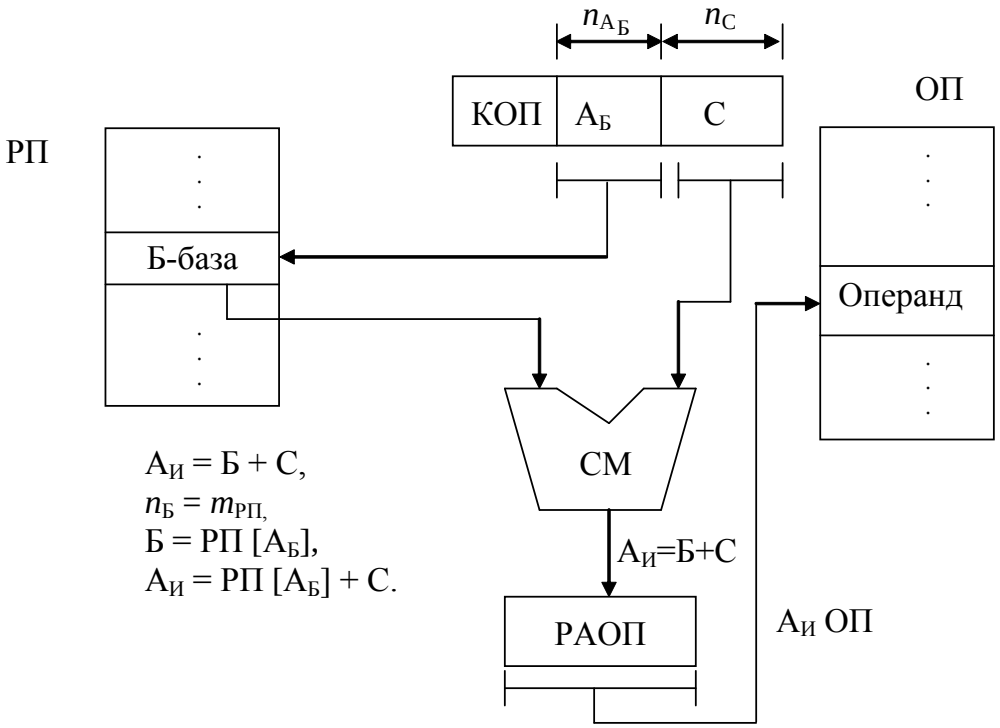


Рис. 2.7.5 Схема формирования относительного адреса способом суммирования кодов базы и смещения: СМ – сумматор; РАОП – регистр адреса ОП; Б – база (базовый адрес); С – смещение; $А_Б$ – адрес регистра базы; $n_Б$ – длина кода базы; $n_С$ – длина поля смещения

Для определения максимальной емкости ОП, адресуемой с помощью базирования способом суммирования, определим длину кода исполнительного адреса

$$n_{А_И} = n_{А_И ОП} = \max\{n_Б; n_С\}.$$

Так как $n_Б = m_{РП}$ и обычно больше, чем $n_С$, то справедливо следующее выражение:

$$M_{ОП} = 2^{n_Б} = 2^{m_{РП}},$$

т.е. максимальная адресуемая емкость ОП определяется разрядностью РП. Длина $n_{А_Б}$ поля кода команды, задающего адрес регистра базы $А_Б$, определяется через емкость РП $M_{РП}$ по формуле

$$n_{А_Б} \geq \log M_{РП}.$$

Таким образом, можно определить количество $n_{А_К}$ двоичных разрядов в адресном поле команды, необходимое для формирования $А_И$ с размещением базы в РП:

$$n_{\text{А}_\text{К}} = n_{\text{А}_\text{Б}} + n_\text{С} = \log_2 M_\text{РП} + n_C .$$

Приведенные выражения позволяют определить числовые значения параметров относительной адресации (базирование способом суммирования).

С помощью метода относительной адресации удается получить так называемый перемещаемый программный модуль, который одинаково выполняется процессором, независимо от адресов, в которых он расположен. При входе в модуль начальный адрес программного модуля (база) загружается в базовый регистр. Все остальные адреса программного модуля формируются через смещение относительно начального адреса (базы) модуля. Таким образом, одна и та же программа может работать с данными, расположенными в любой области памяти, без перемещения данных и без изменения текста программы, только за счет изменения содержания всего одного базового регистра.

Относительная адресация с совмещением составляющих $\text{А}_\text{И}$

Для увеличения емкости адресуемой ОП ($M_\text{ОП}$) без увеличения длины адресного поля команды $n_{\text{А}_\text{К}}$ можно использовать для формирования исполнительного адреса совмещение (конкатенацию) кодов базы и смещения (рис. 2.7.6).

При совмещении кодов базы и смещения

$$n_{\text{А}_\text{И}} = n_\text{Б} + n_\text{С} .$$

Таким образом, $M_\text{ОП} = 2^{n_{\text{А}_\text{И}}} = 2^{n_\text{Б}+n_\text{С}}$.

Следует отметить, что адресное пространство ОП может быть увеличено в $2^{n_\text{С}}$ раз за счет использования способа совмещения. Однако в данном случае начальные адреса массивов не могут быть реализованы произвольно, а должны иметь в младших разрядах $n_\text{С}$ нулей.

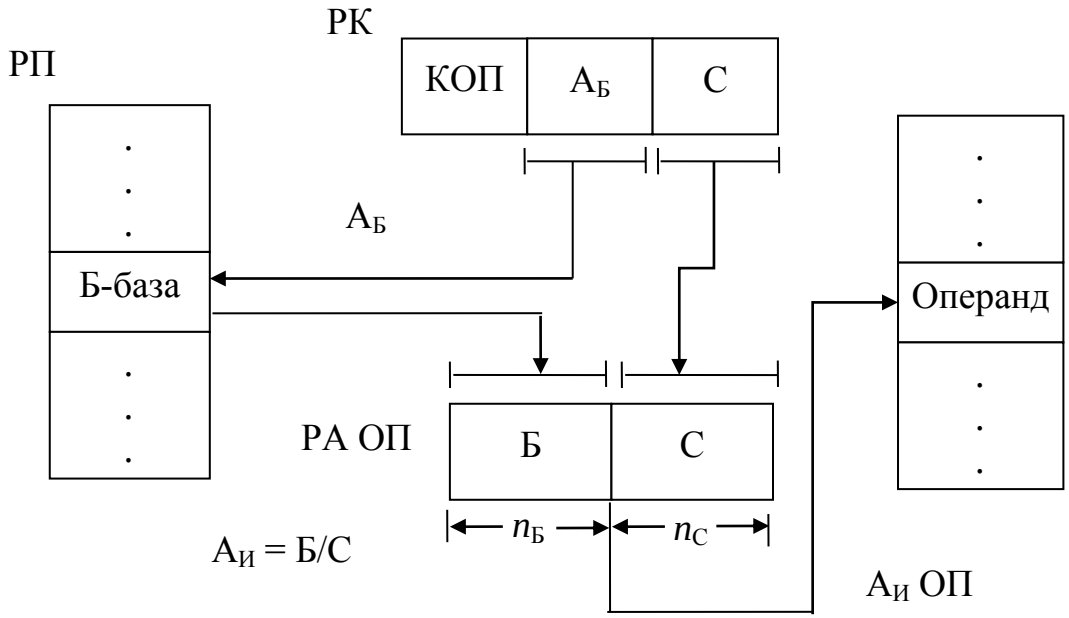


Рис. 2.7.6. Схема формирования относительного адреса способом совмещения кодов базы и смещения

Индексная адресация

Для работы программ с массивами, требующими однотипных операций над элементами массива, удобно использовать индексную адресацию. Схема индексной адресации аналогична базированию путем суммирования (рис. 2.7.7). В этом случае адрес *i*-го операнда в массиве определяется как сумма начального адреса массива (задаваемого полем смещения С) и индекса И, записанного в один из регистров РП и называемым **индексным** регистром. Адрес индексного регистра задается в команде полем адреса индекса – $A_{ИН}$ (аналогично $A_Б$).

В каждом *i*-м цикле содержимое индексного регистра изменяется на величину постоянную (часто равную 1). Использование индексной адресации значительно упрощает программирование циклических алгоритмов.

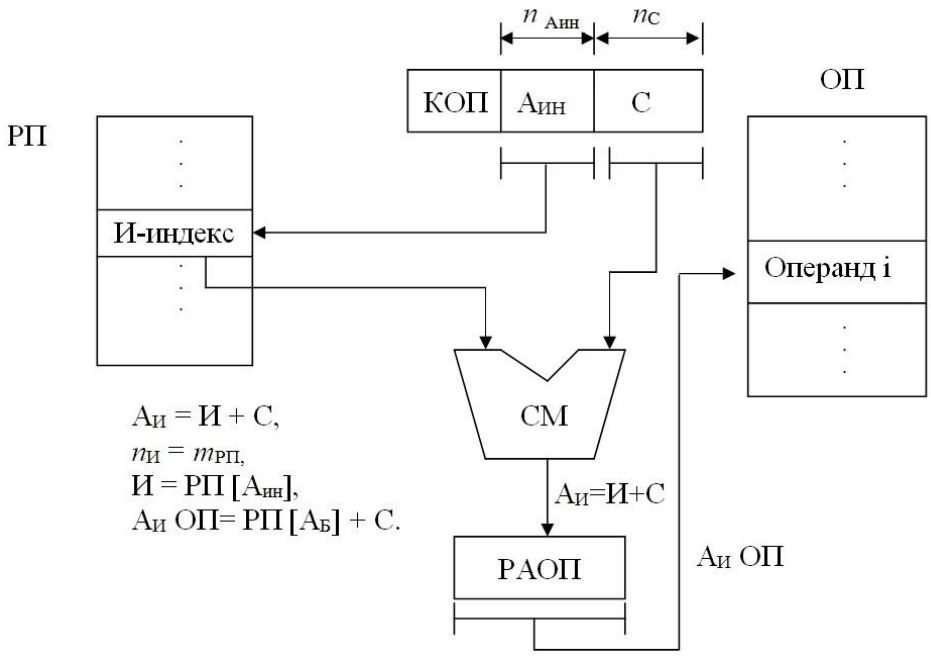


Рис. 2.7.7. Схема индексной адресации

Базово-индексная адресация со смещением

Для эффективной работы при относительной адресации применяется комбинированная индексация с базированием, при которой адрес операнда вычисляется как сумма трех величин (рис. 2.7.8):

$$А_ИОП = Б + И + С.$$

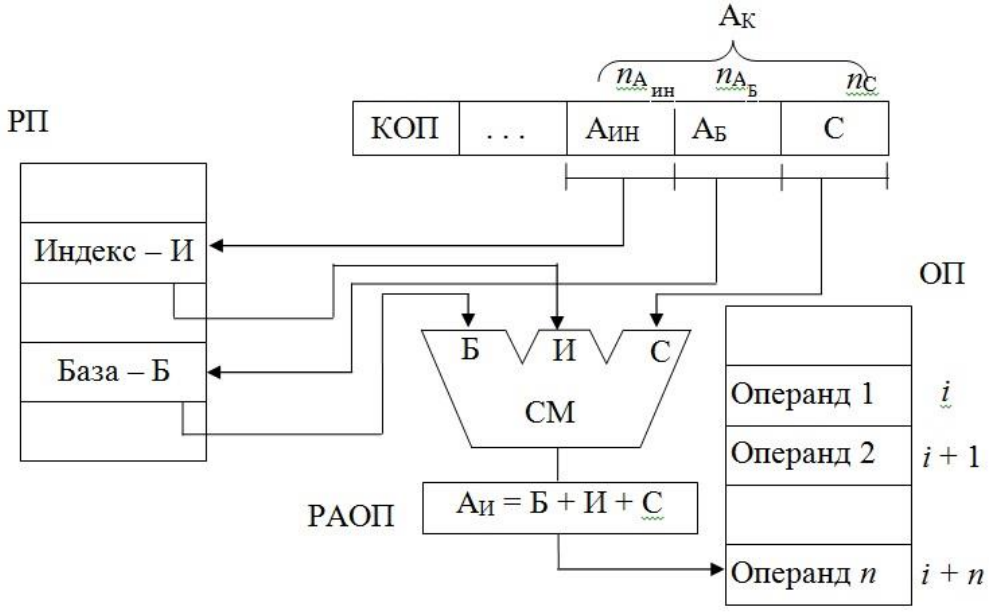


Рис. 2.7.8. Схема формирования исполнительного адреса
при базово-индексной адресации со смещением

2.8 Основные режимы работы x86-64 архитектуры

Как было сказано ранее, корпорацией AMD было разработано 64-разрядное расширение x86-архитектуры, которое получило название «x86-64», т.е. 64-битная x86 (по аналогии с x86-32). Позднее x86-64 архитектура была переименована в AMD64. В отличие от 64-битной архитектуры IA-64, примененной в процессорах Intel Itanium, x86-64 базируется на существующей архитектуре x86-32. Следовательно, процессор, построенный на основе x86-64, может безо всяких проблем исполнять существующие 32-битные приложения, которых написано на текущий момент просто немерено (и в них вложены очень большие деньги). Причем эти приложения могут выполняться без каких бы то ни было потерь в производительности в отличие от того же Intel Itanium, где x86-32 систему команд приходится эмулировать.

В процессорах данной архитектуры существующие в x86 регистры общего назначения расширены с 32 до 64 бит и к ним добавлены еще 8 новых 64-разрядных регистров.

Для реализации одновременной работы как с 32-битным, так и с 64-битным кодом и регистрами архитектура AMD64 предполагает поддержку процессорами двух режимов: **Long Mode («длинный» режим),** имеющего два подрежима – **64-битный режим** и **Compatibility mode (режим совместимости)**, и **Legacy Mode (наследственный режим).** Что они собой представляют, можно понять из табл. 2.8.1.

Таблица 2.8.1. Режимы работы процессора x86-64 архитектуры

Режим		ОС	Необх. перекомпиляция приложений	Характеристики			
				Длина адреса	Длина операнда	Дополнительные регистры	Размер РОН
Long Mode	64-битный	64-битная	ДА	64	64	ДА	64
	Compatibility mode		НЕТ	32	32	НЕТ	32
Legacy Mode		32-бит.	НЕТ	32	32	НЕТ	32
		16-бит.		16	16		

Итак, в **64-битном режиме** обеспечивается поддержка:

- 64-битных виртуальных адресов;
- 8-ми новых и расширенных 64-битных регистров общего назначения;
- 64-битного указателя инструкций RIP;
- сплошного адресного пространства с единым пространством для инструкций, данных и стека.

• 64-битных арифметических и логических операций над целыми числами.

Данный режим снимает ограничение в размерности адресного пространства оперативной памяти, которое в современных 32-разрядных x86 системах составляет 2^{32} = 4 Гбайт.

Для адресации новых регистров в команды введены так называемые «префиксы расширения регистра», кодирование которых осуществляется кодами, использующимися для команд INC <регистр> и DEC <регистр> в 32- и 16-битных режимах. Команды INC и DEC в 64-битном режиме должны кодироваться в более общей, двухбайтовой форме.

Compatibility mode обеспечивает бинарную совместимость с существующими 16- и 32-битными приложениями при работе с 64-битной операционной системой. Этот режим разрешается ОС по принципу отдельных кодовых сегментов. Однако, в отличие от 64-битного режима, сегментация функционирует обычным образом, используя семантику защищенного режима. С точки зрения выполняемого приложения процессор выглядит как обычный x86 центральный процессор (CPU) в защищенном режиме. С точки зрения операционной системы трансляция адресов, работа с прерываниями и исключениями, а также системные структуры данных используют механизмы 64-битного Long Mode.

Наследственный режим (Legacy Mode) обеспечивает бинарную совместимость с 16- и 32-битными операционными системами; полную совместимость с существующими 32-битными реализациями x86 архитектуры, включающей в себя поддержку сегментированной памяти и 32-битных регистров общего назначения и указателя инструкций; процессор уподобляется обычному 32-разрядному x86 CPU. В этом режиме не задействуется ни одна из 64-битных функций.

Нельзя не отметить, что для того, чтобы пользователи смогли воспользоваться преимуществами 64-битного режима, необходим компилятор, который разработан и поставляется вместе с микропроцессором.

Данная архитектура была реализована в процессорах AMD Athlon 64, Opteron.

Особенности архитектуры Intel 64

Архитектура Intel 64 (технология EM64T) в сочетании с соответствующим программным обеспечением поддерживает работу 64-разрядных приложений на серверах, рабочих станциях, настольных ПК и ноутбуках. Она, как и x86-64 от AMD, реализует 64-разрядное расширение регистров, те же режимы работы процессора, ту же программную совме-

стимость с 16- и 32-битными приложениями, а главное – эта технология расширяет адресное пространство виртуальной и физической памяти.

Архитектура Intel 64 поддерживает следующие возможности:

- 64-разрядное сплошное пространство виртуальных адресов;
- 64-разрядные указатели;
- 64-разрядные регистры общего назначения;
- 64-разрядную поддержку вычислений с целыми числами;
- до 1 Тбайт адресного пространства платформы.

2.8.1 Особенности системы команд IA-64

Шестидесятичетырехразрядная интеловская архитектура (IA-64), как было сказано выше, реализует EPIC-концепцию, разработанную совместно фирмами Intel и HP; IA-64 не является 64-разрядным расширением 32-разрядной архитектуры x86 компании Intel или переработкой 64-разрядной архитектуры PA-RISC компании HP; IA-64 представляет собой нечто абсолютно новое – передовую архитектуру, использующую длинные слова команд, предикаты команд, устранение ветвлений, предварительную загрузку данных и другие ухищрения для того, чтобы «извлечь больше параллелизма» из кода программ.

Команды IA-64 можно подразделить на команды работы со стеком регистров (например, **alloc**); целочисленные команды; команды сравнения и работы с предикатами; команды доступа в память; команды перехода; мультимедийные команды; команды пересылок между регистрами; команды выполнения операций над строками и подсчет числа единиц в слове; команды работы с плавающей запятой.

Целочисленные команды IA-64 включают арифметические операции (**add**, **sub** и др.), операции над битами и сдвиги, а также 32-разрядные операции.

Отметим, что команда умножения целых чисел в регистрах общего назначения (GR) отсутствует; для перемножения необходима пересылка целых в регистры с плавающей запятой (FR) и применение операции умножения, выполняемой в функциональном исполнительном устройстве вещественного типа. Некоторые специалисты считают это «наименее удачной» чертой системы команд IA-64.

Команды сравнения и работы с предикатами – это одна из принципиально новых особенностей IA-64 по сравнению с RISC-архитектурой. Приведем несколько типичных примеров команд этой группы. Команда **cmp** сравнивает два регистра GR (или регистр GR и литерал) на одно из 10 возможных условий (больше, меньше или равно и т.п.). Команда **tbit**

тестирует заданный бит GR. Команда **fcmp** сравнивает два числа с плавающей запятой. Однако результатом сравнения является не единственный код условия, что типично для обычных процессоров. Логический результат сравнения (1 – истина, 0 – ложь) записывается обычно в пару предикатных регистров (во второй пишется отрицание первого). Эти значения предикатных регистров (PR) используются затем не только в командах условного перехода, как в обычных микропроцессорах. Почти все команды IA-64 выполнимы «под предикатом», т.е. могут выполняться или нет в зависимости от значения указанного в команде PR-регистра. Это позволяет во многих случаях избежать применения условных переходов, которые, как известно, отрицательно сказываются на производительности процессоров. Вместо этого процессор с архитектурой IA-64, имеющий большое число ресурсов (в частности, регистров и функциональных исполнительных устройств), может исполнять обе ветви программы линейно.

Формат команд IA-64 содержит 41 разряд и имеет фиксированную длину (рис. 2.8.2). Поле КОП занимает 14 разрядов, под адрес 64 предикатных регистров (PR) отводится 6 разрядов, три 7-битных поля (GFR) используются для адресации 128 регистров общего назначения (GR) или регистров с плавающей точкой (FR).

Большинство целочисленных команд трехадресные, а их аргументы находятся в регистрах, однако встречается и литеральное (символьное) представление аргументов. Имеются также модификации команд **add** и **sub**, которые являются четырехадресными: в них к сумме/разности регистров прибавляется/вычитается 1.

Рис. 2.8.2. Формат инструкций IA-64

Команды в формате IA-64 упакованы по три в 128-битный LIW (long instruction word) – пакет (рис. 2.8.3).

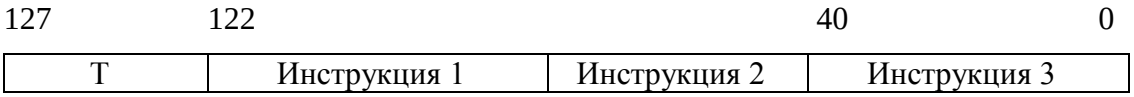


Рис. 2.8.3. Пакет инструкций IA-64

В каждый пакет при трансляции компилятор помещает шаблон, который размещается в 5-битовом поле T. Шаблон пакета указывает не только на то, какие команды в пакете могут выполняться независимо, но и какие команды из следующего пакета могут выполняться параллель-

но. Команды в пакетах не обязательно должны быть расположены в том же порядке, что и в машинном коде, и могут принадлежать к различным путям ветвления. Компилятор может также помещать в один пакет зависимые и независимые команды, поскольку возможность параллельного выполнения определяется шаблоном пакета. В отличие от некоторых ранее существовавших архитектур со сверх длинными словами команд (VLIW) IA-64 не добавляет команд «нет операции» (NOPS) для дополнения пакетов.

2.9 Регистровые структуры центрального процессора

2.9.1 Регистровые структуры процессоров IA-32

В процессорах IA-32 можно выделить следующие группы регистров:

1. Основные функциональные регистры:
- регистры общего назначения (GPR);
- указатель команд;
- регистр флагов;
- регистры сегментов.

2. Регистры процессора обработки чисел с плавающей точкой (FPU):
- регистры данных;
- регистр тегов;
- регистр состояния;
- регистр указателей команд и данных FPU;
- регистр управления FPU.

3. Векторные регистры расширений SSE.

4. Системные регистры:
- регистры управления микропроцессора;
- регистры системных адресов.

5. Регистры отладки и тестирования.

Регистры первых трех групп используются при выполнении прикладных программ, четвертой группы – системных операций, пятой – при отладке и тестировании. Все группы регистров x86 отражены на рис. 2.9.7.

Регистры общего назначения

Восемь 32-разрядных регистров (EAX, ECX, EDX, EBS, EBP, ESP, ESI, EDI) предназначены для хранения данных и адресов. Они поддерживают работу с данными разрядностью 1, 8, 16 и 32 бита, битовыми

полями длиной от 1 до 32 бит и адресами размером 16 и 32 бита. Младшие 16 разрядов этих регистров (рис.2.9.1) доступны отдельно при использовании соответствующего имени, например регистр EAX (имя AX для 16 разрядов).

При операциях с байтами можно отдельно обращаться к младшему байту (разряды 0–7) и старшему байту (8–15) по именам AL и AH. Доступ к отдельным байтам обеспечивает дополнительную гибкость при операциях с данными.

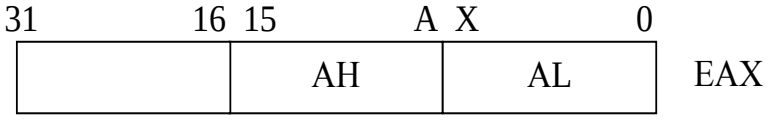


Рис. 2.9.1 Структура регистра общего назначения EAX

Регистры сегментов и дескрипторы сегментов

Шесть 16-разрядных сегментных регистров (CS, SS, DS, ES, FS, GS) содержат значения селекторов сегментов, указывающих на текущие адресуемые сегменты памяти. С каждым из них связан программно-недоступный регистр дескриптора сегмента (рис. 2.9.2).

В защищенном режиме каждый сегмент может иметь размер от 1 байта до 4 Гбайт, в режиме реальных адресов максимальный размер сегмента составляет 64 Кбайта.

Селектор в CS обеспечивает обращение к текущему сегменту команд, селектор в SS – к текущему сегменту стека, селекторы в DS, ES, FS, GS – к текущим сегментам данных. Каждый регистр дескриптора содержит базовый адрес сегмента, 32-разрядный размер сегмента и другие необходимые атрибуты.

Регистры сегментов

15 0

Селектор	CS
Селектор	SS
Селектор	DS
Селектор	ES
Селектор	FS
Селектор	GS

Регистры дескрипторов

Базовый адрес	Размер сегмента	Другие атрибуты				

Рис. 2.9.2 Регистры сегментов и соответствующие регистры дескрипторов

Когда в регистр сегмента загружается новое значение селектора, содержимое соответствующего регистра дескриптора автоматически корректируется. В реальном режиме базовый адрес сегмента получается путем сдвига значения селектора на 4 разряда влево (20 разрядов), максимальный размер и атрибуты сегмента в реальном режиме имеют фиксированные значения.

Указатель команд

Указатель команд (рис. 2.9.3) представляет собой 32-разрядный регистр с именем EIP, содержимое которого используется в качестве смещения при определении адреса следующей выполняемой команды. Смещение задается относительно базового адреса сегмента команд CS. Младшие 16 бит (0–15) содержат 16-разрядный указатель команд с именем IP, который используется при 16-разрядной адресации.

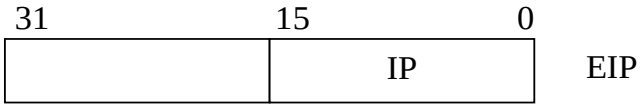


Рис. 2.9.3. Структура регистра указателя команд

Указатель команд непосредственно программисту недоступен. Его содержимое изменяется при выполнении команд передачи управления и прерываний.

Регистр флагов

Регистр флагов является 32-разрядным, имеет имя EFLAGS. Его разряды содержат признаки результата выполнения команды, управляют обработкой прерываний, последовательностью вызываемых задач, вводом/выводом и рядом других процедур.

Регистры процессора обработки чисел с плавающей точкой

Набор регистров, входящих в блок FPU, изображен на рис. 2.9.4

При работе FPU 80-разрядные регистры ST0–ST7 образуют кольцевой стек, в котором хранятся числа с плавающей точкой, представленные в формате с расширенной точностью.

Рис. 2.9.4. Регистры блока FPU

Регистр тегов FPU содержит 16-разрядное слово, включающее восемь двухбитовых тегов. Каждый тег (признак) характеризует содержимое одного из регистров данных.

Тег определяет, является ли регистр пустым (незаполненным) – код 11, или в него введено конечное число – 00 (достоверное значение), или нуль – 01, неопределенное значение (бесконечность) – 10 (нет числа и неподдерживаемый формат). Слово тегов позволяет оптимизировать функционирование FPU посредством идентификации пустых и непустых регистров данных, проверить содержимое регистра без сложного декодирования хранящихся в нем данных.

Регистры MMX-технологии

При реализации MMX-команд регистры данных FPU используются как 64-разрядные регистры MM0–MM7 (см. рис. 2.9.4), где могут храниться несколько целочисленных операндов (восемь 8-разрядных, четыре 16-разрядных, два 32-разрядных или один 64-разрядный), над которыми одновременно выполняется поступившая в процессор команда.

Векторные регистры SSE расширений

Потоковые команды расширений SSE используют восемь 128-разрядных регистров XMM0–XMM7 (16 регистров XMMXMM15 в x86-64), в которых могут храниться несколько вещественных или целочисленных операндов (рис. 2.9.7).

Системные регистры

Системные регистры управляют функционированием микропроцессора в целом и режимами работы отдельных внутренних блоков: процессора с плавающей точкой, кэш-памятью, диспетчера памяти. Эти регистры доступны только в защищенном режиме для программ.

Набор системных регистров включает регистры управления, регистры системных адресов и сегментов.

Регистры управления 32-разрядные, служат для фиксации общего состояния процессора. Эти регистры вместе с регистрами системных адресов хранят информацию о состоянии процессора, которое затрагивает все задачи.

Регистры отладки и тестирования

Набор программно-доступных регистров поддерживает отладку программ и тестирования внутренних блоков процессора. Встроенный алгоритм самотестирования (BIST) осуществляет поиск ошибки в микрокоде и в больших логических матрицах, а также тестирование кэш-памяти команд и данных, буферов ассоциативной трансляции (TLB) и устройств постоянной памяти. Внутренние счетчики контролируют работу процессора и проводят подсчет событий. Введена новая функция – мониторинг термического состояния системной платы.

Переименование регистров

В современных процессорах используются блоки регистров замещения (регистровые файлы) для целочисленных, вещественных и векторных данных. Для любого указанного в команде логического регистра (программно можно обращаться в x86 только к 8-ми регистрам общего назначения GPR, 8-ми регистрам с плавающей точкой ST или 8-ми MMX / XMM-регистрам) выделяется один из физических регистров соответствующего блока регистров замещения, содержащего, например, 128 регистров. Эта процедура (переименование регистров) позволяет увеличить количество используемых регистров процессора, а также позволяет выполнять команды, в которых задействованы одни и те же логические регистры, одновременно или с изменением их последовательности.

2.9.2 Регистровые структуры процессоров AMD64 (Intel64)

В процессорах x86-64 (AMD64), Intel64 архитектур (рис. 2.9.5) существующие в x86 регистры общего назначения (GPR) расширены с 32

до 64 бит (RAX, RBX, RCX, RDX, RBP, RSP, RSI, RDI) и к ним добавлены еще 8 новых 64-разрядных регистров (R8–R15). Также 8 новых 128- битных регистров (XMM8–XMM15) добавлено в блок SSE, что обеспечивает поддержку SSE2.

В блоке FPU используются существующие в x87 регистры данных ST0–ST7 (80-разрядные) и 64-разрядные мультимедийные регистры MM0–MM7, объединенные в общее пространство с регистрами ST.

Регистр указателя команд (RIP) и регистр флагов (RFLAGS) также расширены до 64 разрядов.

Векторные регистры AVX расширений

Инструкции расширений AVX, AVX2 используют шестнадцать 256-битных регистров YMM0–YMM15, в которых могут храниться несколько целочисленных или с плавающей запятой операндов.

Векторные регистры MIC расширений

Инструкции расширений MIC используют шестнадцать 512-битных регистров, в которых могут храниться несколько целочисленных или с плавающей запятой операндов.

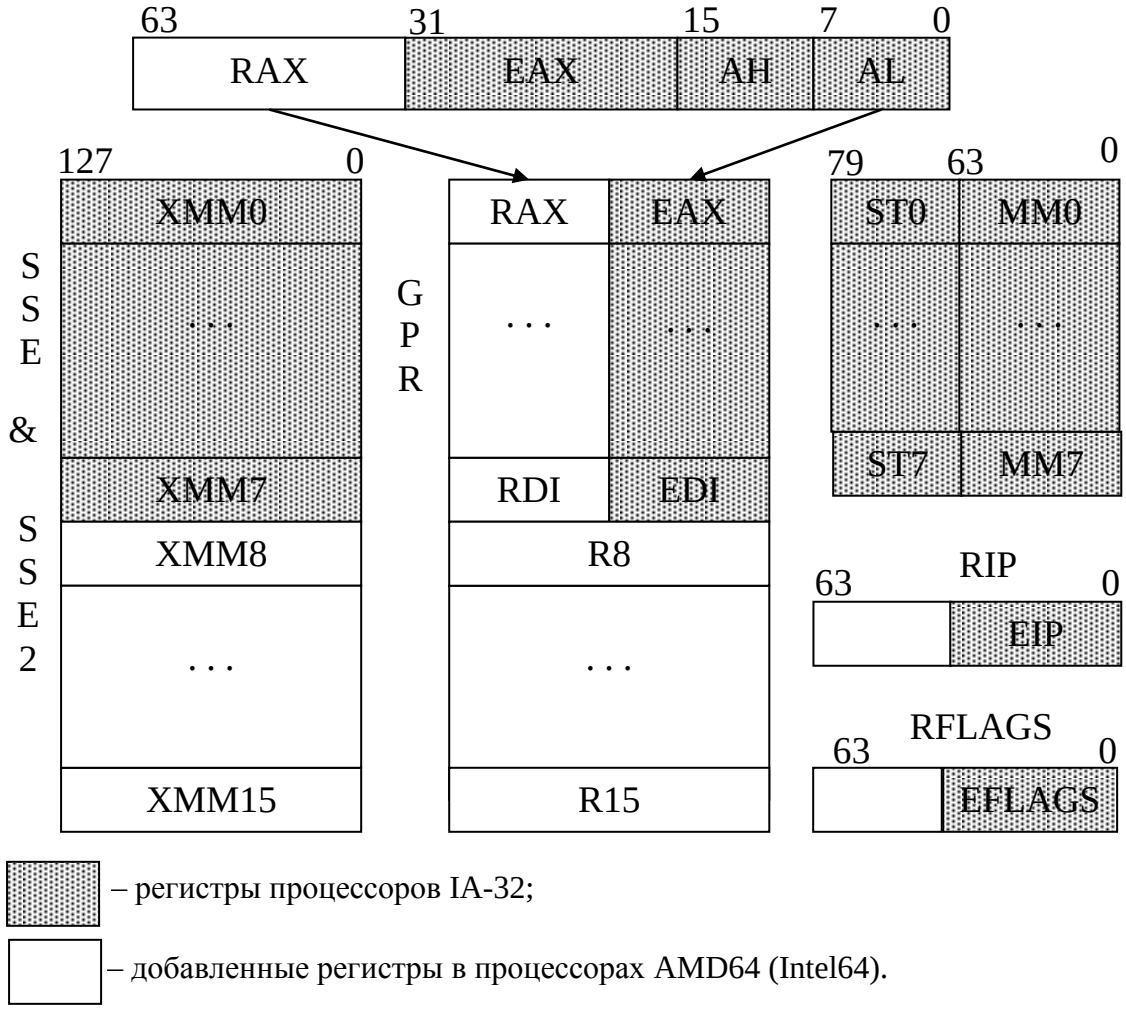


Рис. 2.9.5 Регистровые структуры процессоров AMD64 (Intel64)

Регистровые структуры процессоров IA-64

В состав регистровых файлов IA-64 (рис.2.9.6) входят: 128 регистров общего назначения GPR (64-разрядных); 128 регистров с плавающей запятой FR (82-разрядных); 128 прикладных регистров (в основном 64-разрядных) AR; 64 одноразрядных регистра предикатов PR; 8 регистров переходов BR (64-разрядных); не менее 4-х регистров идентификатора процесса CPUID; счетчик команд IP; регистр маркера текущего окна CFM стека регистров и др.

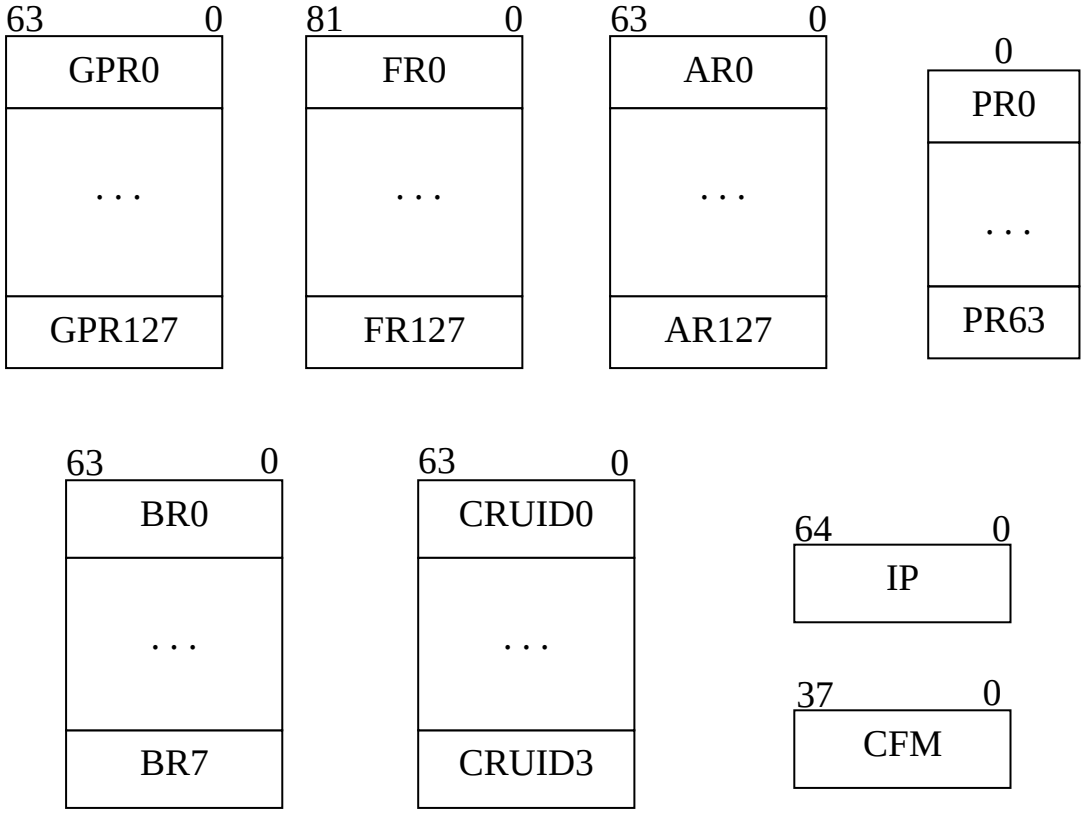


Рис. 2.9.6. Регистровые структуры процессоров IA-64

64-разрядные регистры GPR0–GPR127 применяются не только для целочисленных операций IA-64; GPR8–GPR31 в режиме IA-32 используются также под целочисленные регистры и регистры селекторов и дескрипторов сегментов IA-32. GPR0–GPR31 называются **статическими регистрами** (GPR0 всегда содержит 0), а GPR32–GPR127 – **стекируемыми регистрами**. Статические регистры «видны» всем программам. Стекируемые регистры становятся доступными в программной единице через окно стека регистров, включающее локальные и выходные регистры, число которых задается командой alloc.

82-разрядные регистры с плавающей запятой FR0–FR127 также подразделяются на **статические** (FR0–FR31, причем всегда FR0 = 0.0, FR1 = 1.0) и **вращаемые** (FR32–FR127). FR8–FR31 в режиме IA-32 содержат числа с плавающей запятой и мультимедийные регистры. Вращение регистров является в некотором роде частным случаем переименования регистров, применяемым в современных суперскалярных процессорах с внеочередным выполнением команд. В отличие от них (пе-

реименование регистров осуществляется аппаратно) вращение регистров в IA-64 управляется программно.

Прикладные регистры AR0–AR127 – специализированные. Ряд AR-регистров является фактически регистрами IA-32; AR0–AR7 называются регистрами ядра. Запись в них привилегированна, но они доступны на чтение в любом приложении и используются для передачи приложению сообщений от операционной системы. AR16 (RSC) – регистр конфигурации стека регистров, используемый для управления работой стека регистров IA-64; AR40 (FPSR) – регистр состояния для команд с плавающей запятой IA-64.

Регистры предикатов PR0–PR63 являются одноразрядными, в них помещаются результаты выполнения команд сравнения. Обычно эти команды устанавливают сразу два соседних регистра PR в состояния «1» – истина, «0» – ложь или, наоборот, в зависимости от значения условия. Такая избыточность обеспечивает дополнительную гибкость.

64-разрядные регистры переходов BR0–BR7 применяются для указания адреса перехода в соответствующих командах перехода (если адрес перехода не кодируется в команде явно).

В регистрах CPUID 0 и CPUID 1 находится информация о производителе, в регистре CPUID 2 – серийный номер процессора, а в регистре CPUID 3 задается тип процессора (семейство, модель, версия архитектуры и т.п.) и число CPUID-регистров. Разряды регистра CPUID4 указывают на поддержку конкретных особенностей IA-64, которые реализованы в данном процессоре.

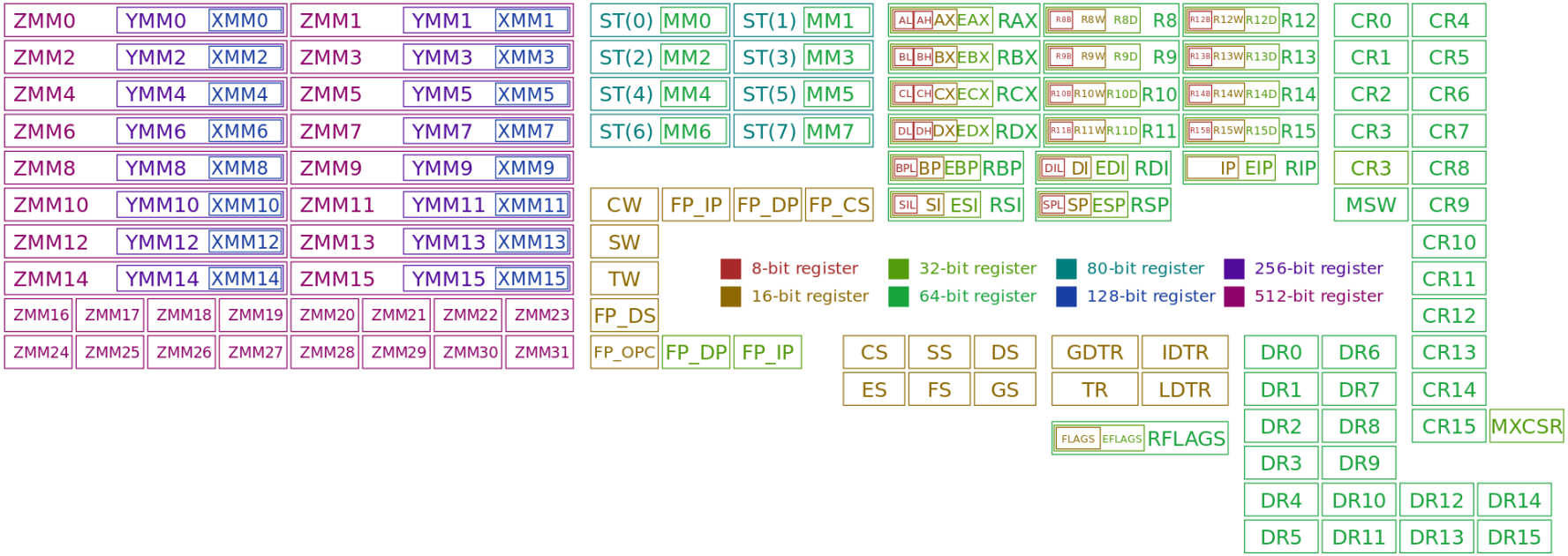


Рис. 2.9.7. Регистровые структуры процессоров x86

2.10 Принципы организации системы прерывания программ

Во время выполнения ЭВМ текущей программы внутри машины и в связанной с ней внешней среде (технологический процесс, управляемый ЭВМ) могут возникать события, требующие немедленной реакции на них со стороны машины.

Реакция состоит в том, что при поступлении сигнала на прерывание выполнение текущей последовательности команд приостанавливается и управление передается обработчику прерываний, который реагирует на события и обслуживает его, после чего возвращает управление в прерванный код программы.

Рассматриваемый процесс, называемый прерыванием программ, поясняется на рис. 2.10.1.

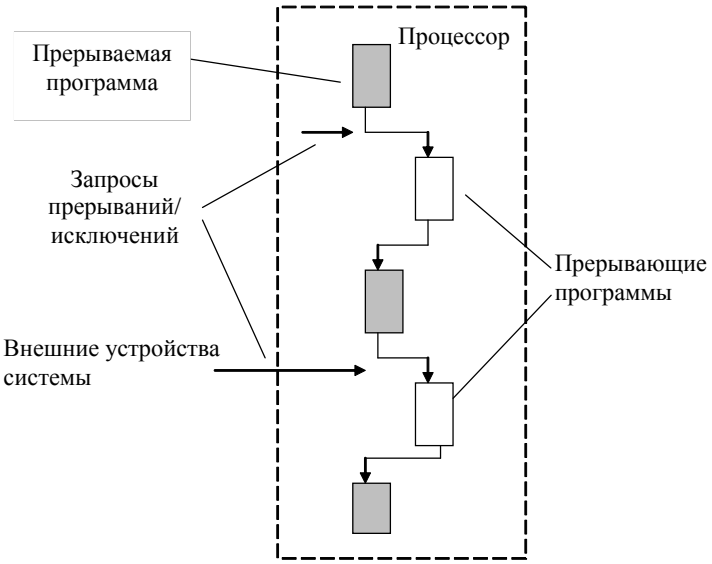


Рис. 2.10.1. Процесс прерывания программы

Обработчик прерываний – это программа обработки прерываний, являющаяся частью ОС, предназначенная для выполнения ответных действий на условие вызвавшее прерывание.

Типы прерываний

Прерывания, возникающие при работе вычислительной машины, можно разделить на несколько типов.

Аппаратные прерывания вызываются физическими устройствами и возникают по отношению к программе асинхронно, т.е. в общем

случае невозможно предсказать, когда и по какой причине программа будет прервана.

Аппаратные прерывания не координируются с работой программного обеспечения. Когда вызывается прерывание, то процессор прерывает свою работу, выполняет прерывание, а затем возвращается на прежнее место.

Внешние прерывания возникают по сигналу какого-либо внешнего устройства, например:

- прерывание, которое информирует систему о том, что требуемый сектор диска уже прочитан, его содержимое доступно программе;
- прерывание, которое информирует систему о том, что завершилась печать символа на принтере и необходимо выдать следующий символ;
- прерывания по нарушению питания;
- нормальное завершение некоторой операции ввода-вывода (нажатие клавиши на клавиатуре);
- прерывание по таймеру.

Внутренние прерывания вызываются событиями, которые связаны с работой процессора и являются синхронными с его операциями, а именно прерывание происходит:

- при нарушении адресации (в адресной части выполняемой команды указан запрещенный или несуществующий адрес, обращение к отсутствующему сегменту или странице при организации механизмов виртуальной памяти);
- при наличии в поле кода незадействованной двоичной комбинации;
- при делении на ноль;
- при переполнении или исчезновении порядка;
- при обнаружении ошибок четности, ошибок в работе различных устройств аппаратуры средствами контроля.

Запросы, возникающие в таких (исключительных) ситуациях, называются **исключениями.**

Исключения делятся на **отказы** и **выходы из процесса**, в зависимости от способа сообщения о них и возможности перезапуска процессора с вызвавшей их команды. **Отказы** – это исключения, которые выявляются и обслуживаются перед выполнением команды. Они могут иметь место в виртуальной системе памяти, когда процессор обращается к несуществующей странице или сегменту. В процессе обработки такого исключения операционная система обращается к странице или сегменту на диске, а процессор перезапускает команду. **Выход из про-**

цесса является исключением, которое не позволяет точно локализовать причину, вызвавшую исключительную ситуацию. Выходы из процесса используются для сообщения о крупных ошибках, таких как неисправности аппаратуры или ошибки в системных таблицах. Адресом возврата из процедуры обработки исключений всегда является команда, вызвавшая исключение, возможны префиксы.

Программные прерывания не являются асинхронными. Программы могут сами вызывать прерывания с заданным номером. Для этого они используют команду INT. По этой команде процессор осуществляет практически те же действия, что и при обычных прерываниях, но только это происходит в предсказуемой точке программы. Там, где программист поместил данную команду.

Программные прерывания в прямом смысле прерываниями не являются, поскольку представляют собой лишь специфический способ вызова процедур не по адресу, а номеру в таблице.

Механизм программных прерываний был специально введен для того, чтобы:

- переключение на системные программные модули происходило не просто как переход в подпрограмму, а точно таким же образом, как и обычные прерывания. Этим обеспечивается автоматическое переключение процессора в привилегированный режим с возможностью исполнения любых команд;
- использование программных прерываний приводит к более компактному коду программ по сравнению с использованием стандартных команд выполнения процедур.

В дальнейшем обработку всех запросов, прерывающих ход выполнения текущей программы, будем называть просто **прерываниями.**

Программу, затребованную запросом прерывания, назовем **прерывающей программой,** противопоставляя ее **прерываемой программе,** выполнявшейся в ЭВМ до появления запроса.

Характеристики системы прерывания

Для оценки эффективности систем прерывания могут быть использованы следующие характеристики:

1. **Общее число запросов прерывания** (входов в систему прерывания).

2. Время реакции – время между появлением запроса прерывания и моментом прерывания текущей программы. На рис. 2.10.1 приведена упрощенная временная диаграмма процесса прерывания.

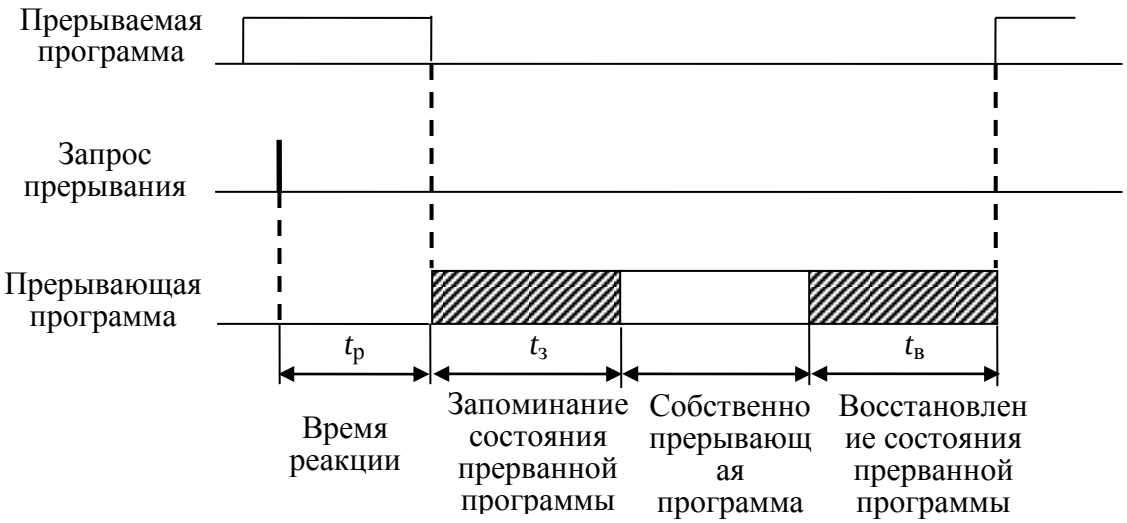


Рис. 2.10.1 Упрощенная временная диаграмма процесса прерывания

Для одного и того же запроса задержки в исполнении прерывающей программы зависят от того, сколько программ со старшим приоритетом ждут обслуживания, поэтому время реакции определяют для запроса с наивысшим приоритетом ($t_Р$).

Время реакции зависит от того, в какой момент допустимо прерывание. Большей частью прерывание допускается после окончания текущей команды. В этом случае время реакции определяется в основном длительностью выполнения команды.

Это время реакции может оказаться недопустимо большим для ЭВМ, предназначенных для работы в реальном масштабе времени. В таких машинах часто допускается прерывание после любого такта выполнения команды (микрокоманды). Однако при этом возрастает количество информации, подлежащей запоминанию и восстановлению при переключении программ, т.к. в этом случае необходимо сохранять также и состояние счетчика тактов, регистра кода операции и некоторых других узлов. Такая организация прерывания возможна только в машинах с быстродействующей сверхоперативной памятью.

Имеются ситуации, в которых желательно немедленное прерывание. Если аппаратура контроля обнаружила ошибку, то целесообразно сразу же прервать операцию, пока ошибка не оказала влияние на следующие такты работы программы.

3. Затраты времени на переключение программ (издержки прерывания) равны суммарному расходу времени на запоминание и восстановление состояния программы (рис. 2.10.1):

$$t_{изд} = t_з + t_в.$$

4. Глубина прерывания – максимальное число программ, которые могут прерывать друг друга. На рис. 2.10.2 показаны процессы прерывания в системах с различной глубиной прерывания (предполагается, что приоритет каждого последующего запроса выше предыдущего). Если после перехода к прерывающей программе и вплоть до ее окончания прием запросов прекращается, то говорят, что система имеет глубину прерывания, равную 1 (относительная дисциплина обслуживания).

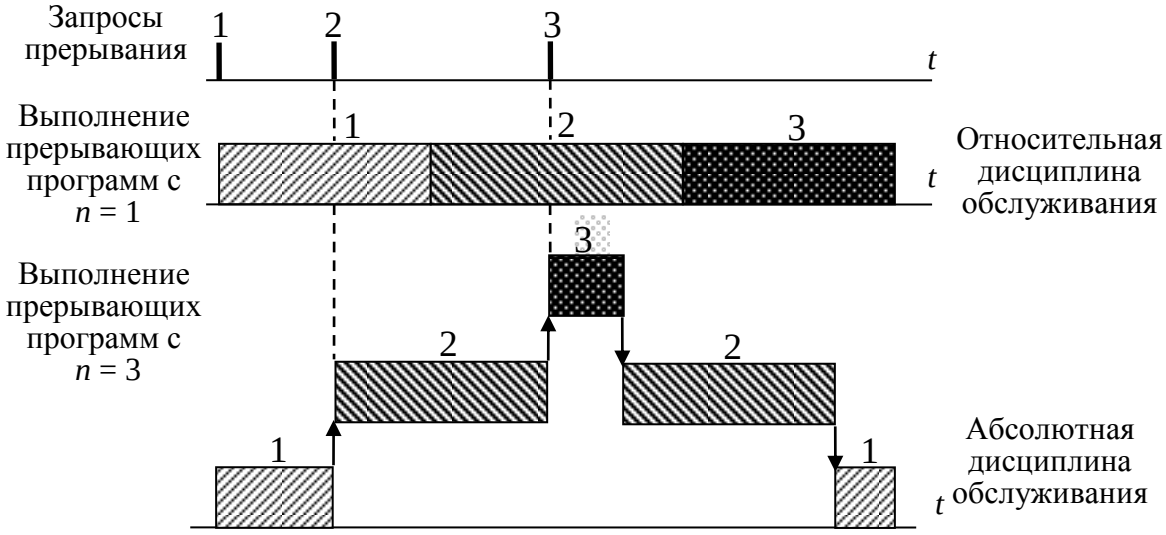


Рис. 2.10.2. Процессы прерывания с различной глубиной прерывания и дисциплиной обслуживания

Глубина равна *n*, если допускается последовательное прерывание до *n* прерывающих программ (абсолютная дисциплина обслуживания). Глубина прерывания обычно совпадает с числом уровней приоритета в системе прерывания. Система с большим значением глубины прерывания обеспечивает более быструю реакцию на срочные запросы.

Если запрос окажется не обслуженным к моменту прихода нового запроса от того же источника, то возникает так называемое **насыщение системы прерывания**. В этом случае предыдущий запрос от данного источника будет машинально утерян, что недопустимо. Существуют специальные методы борьбы с этой ситуацией.

Программно-управляемый приоритет прерывающих программ

Относительная степень важности программ, их частота повторения, относительная степень срочности в ходе вычислительного процесса мо-

гут меняться, требуя установления новых приоритетных отношений. Поэтому во многих случаях приоритет между прерывающими программами не может быть зафиксирован раз и навсегда. Необходимо иметь возможность изменять, по мере необходимости, приоритетные соотношения программным путем. Приоритет между прерывающими программами должен быть динамичным, т.е. программно-управляемым.

В ЭВМ широко применяется способ маскирования прерываний.

Существуют два специальных внешних сигнала среди входных сигналов процессора, при помощи которых можно прервать выполнение текущей программы и тем самым переключить работу центрального процессора. Это сигналы NMI (Non Mascable Interrupt, немаскируемое прерывание) INTR (INTerrupt Request, запрос на прерывание).

Соответственно внешние прерывания подразделяются на два вида: немаскируемые и маскируемые.

Маска прерывания представляет собой двоичный код, разряды которого поставлены в соответствие запросам или классам (уровням) прерывания. Маска загружается командой программы в регистр маски (рис. 2.10.3).

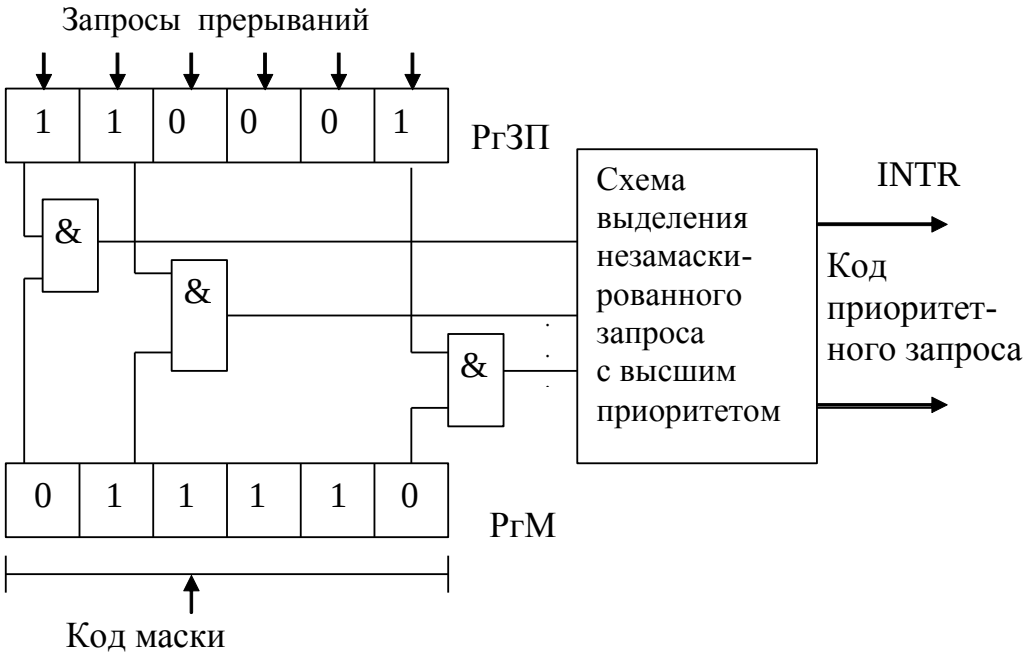


Рис. 2.10.3. Маскирование прерываний

Состояние 1 в данном разряде регистра маски (РгМ) разрешает, а состояние 0 запрещает (маскирует) прерывание текущей программы от соответствующего запроса в регистре запросов прерываний (РгЗП). Таким образом, программа, изменяя маску в регистре маски, может

устанавливать произвольные приоритетные соотношения между программами без перекоммутации линий, по которым поступают запросы прерывания. Каждая прерывающая программа может установить свою маску. При формировании маски 1 устанавливаются в разряды, соответствующие запросам (прерывающим программам) с более высоким, чем у данной программы, приоритетом. Схемы И выделяют поступившие незамаскированные запросы прерывания, из которых специальная схема выделяет наиболее приоритетный запрос, формирует код его номера и вырабатывает основной сигнал прерывания (INTR).

С замаскированным запросом, в зависимости от причины прерывания, поступают двояким образом: или он игнорируется, или запоминается, с тем, чтобы осуществить затребованные действия, когда запрет будет снят. Например, если прерывание вызвано окончанием операции в ПУ, то его следует, как правило, запомнить, так как иначе ЭВМ останется неосведомленной о том, что ПУ освободилось.

Возможность прерывания программ – важное архитектурное свойство ЭВМ, позволяющее эффективно использовать производительность процессора при наличии нескольких процессов, протекающих параллельно во времени, требующих в произвольные моменты времени управления и обслуживания со стороны процессора.

Чтобы ЭВМ могла, не требуя больших усилий от программиста, реализовывать с высоким быстродействием прерывания программ, машине необходимо придать соответствующие аппаратурные и программные средства, совокупность которых получила название **системы прерывания программ.** В качестве аппаратных средств используется **контроллер прерывания** (блок прерывания).

Основными функциями системы прерывания являются:

- запоминание состояния прерываемой программы и осуществление перехода к прерывающей программе;
- восстановление состояния прерванной программы и возврат к ней.

При наличии нескольких источников запросов прерывания между ними должны быть установлены **приоритетные соотношения,** определяющие, какой из нескольких поступивших запросов подлежит обработке в первую очередь, и **устанавливающие**: имеет право или нет данный запрос (прерывающая программа) прерывать ту или иную программу.

Структура аппаратных средств системы прерывания

В процессе эволюции компьютеров на базе Intel, эволюционировала и архитектура обработки прерываний. Изначально контроллер прерываний **PIC (Programmable Interrupt Controller)** состоял из одной

микросхемы Intel 8259 с 8-ю входными линиями от устройств и двумя выходными сигнальными линиями к процессору — INT и NMI. Кроме того контроллер прерываний связан с процессором через шину. Контроллер вызывает внимание процессора выставлением сигнала INT или NMI, а номер возникшего прерывания передает через шину. Кроме того, через шину процессор может осуществлять управление контроллером. На физическом уровне различают даже несколько шин, через которые связаны контроллер с процессором.

Восемь входных линий контроллера прерываний позволяли назначать устройствам 8 разных аппаратных прерываний. Позже этого оказалось не достаточно. Чтобы расширить количество входных линий в IRQ два одинаковых контроллера связали каскадно. Выходная сигнальная линия дополнительного контроллера завязывалась на входную линию главного контроллера с номером 2. Таким образом, для устройств в такой совместной схеме осталось 15 входных линий.

С появлением многоядерных систем потребовались дополнительные контроллеры прерываний. Интелу потребовалось решать две задачи. Во-первых, снова появилась нехватка входных линий прерываний. Во-вторых, потребовался механизм межпроцессорной синхронизации в многопроцессорных системах. На смену PIC, Intel разработал APIC.

APIC (Advanced Programmable Interrupt Controller) — улучшенный программируемый контроллер прерываний.

Преимущества расширенного контроллера прерываний:

- возможность реализации межпроцессорных прерываний — сигналов от одного процессора другому;
- поддержка до 256 входов IRQ, в отличие от 1 5 в классической архитектуре;
- крайне быстрый доступ к регистрам текущего приоритета прерывания и подтверждения прерывания.

APIC поддерживался в ОС Windows, начиная с Windows NT 4.0.

APIC состоит из двух модулей:

- **LOCAL APIC** — располагается в ядре процессора, если система многоядерна - в каждом ядре;
- **I/O APIC** — контроллер, расположенный на системной плате, обычно как часть микросхем обрамления.

На примере структуры соединения контроллеров прерывания (рис. 2.10.4) внутрь каждого ядра процессора поместили контроллер с именем Local APIC, а в дополнение к 2-м контроллерам 8259, добавили IOAPIC. Появилась дополнительная шина APIC, связывающая между собой IOAPIC и LocalAPIC. Появились линии синхронизации процессоров.

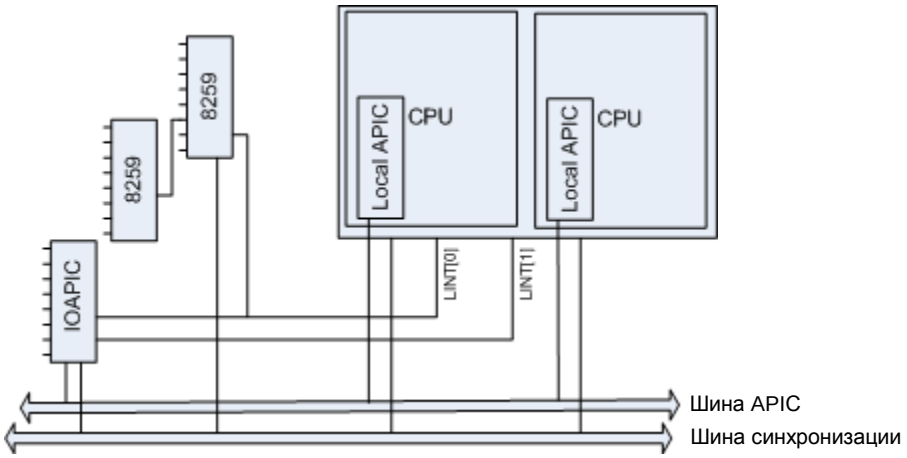


Рис. 2.10.4 Пример структуры соединения контроллеров прерывания

В настоящий момент наблюдается тенденция к отказу от IO APIC, как и проводников IRQ, и переходу на Message Signaled Interrupts.

Message Signaled Interrupts (MSI – прерывания, инициируемые сообщениями) — альтернативная форма прерываний, доступная в PCI версии 2.2 и более поздних, PCI-X, а также обязательная в PCI Express любых версий. Вместо присваивания фиксированного номера запроса на прерывание, устройству разрешается записывать сообщение по определённому адресу системной памяти, на деле отображенному на аппаратуру локального контроллера прерываний (local APIC) процессора. Для записи сообщения используется тот же механизм захвата шины (bus mastering), что и для DMA (Direct Memory Access – прямой доступ к памяти).

Для записи сообщений каждое устройство, использующее MSI может иметь от одной до тридцати двух уникальных областей памяти.

Все прерывания шины PCI Express всегда доставляются как MSI, даже при использовании эмуляции традиционных номеров проводников прерываний.

Достоинства MSI:

- возможность полного отказа от проводников IRQ от устройств и разъемов PCI до главного контроллера прерываний (IO APIC), а также от самого главного контроллера прерываний, что упрощает материнскую плату;
- в многопроцессорных и многоядерных системах устройства, использующие несколько областей MSI, получают возможность самостоятельно выбирать процессор/ядро для обработки конкретного прерывания, причем делать это полностью на уровне аппаратуры без исполнения

программного кода. Это позволяет оптимизировать работу путем размещения большей части структур драйвера устройства и связанного с ним программного обеспечения (сетевых протоколов и т.д.) в кэше конкретного процессора.

MSI поддерживается в операционных системах Microsoft Windows Vista и более поздних, в ОС FreeBSD с версии 6.3, а также в ядре Linux начиная с версии 2.6.8.

Организация перехода к прерывающей программе

Наиболее гибким и динамичным является **векторное прерывание,** при котором вектор начального состояния прерывающей программы называют **вектором прерывания.** Он содержит всю необходимую информацию для перехода к прерывающей программе, в том числе ее начальный адрес. Каждому запросу (уровню) прерывания соответствует свой вектор прерывания, способный инициировать выполнение соответствующей прерывающей программы. Векторы прерывания обычно находятся в специально выделенных фиксированных ячейках памяти (стеке).

Главное место в процедуре перехода к прерывающей программе занимает передача из соответствующего регистра (регистров) процессора в память (стек) на сохранение текущего вектора состояния прерываемой программы (чтобы можно было вернуться к ее исполнению) и загрузка в регистр (регистры) процессора вектора прерывания прерывающей программы, к которой при этом переходит управление процессором. Источник прерывания, выставляя запрос прерывания, посылает в процессор (выставляет на шины интерфейса) код адреса в памяти своего вектора прерывания.

При векторном прерывании каждому запросу прерывания, или, другими словами, устройству – источнику прерывания, соответствует переход к начальному адресу соответствующей прерывающей программы, задаваемому вектором прерывания.

2.11 Структурная организация современных универсальных микропроцессоров

Характерными чертами современных универсальных микропроцессоров являются:

1. Суперскалярная архитектура, обеспечивающая одновременное выполнение нескольких команд в параллельно работающих исполнительных устройствах.

2. Динамическое изменение последовательности команд (выполнение команд с опережением – спекулятивное выполнение).

3. Конвейерное исполнение команд.
4. Предсказание направления ветвлений.
5. Предварительная выборка команд и данных.
6. Параллельная обработка потоков данных.
7. Многоядерная структура.
8. Многопотоковая обработка команд.
9. Пониженное энергопотребление.

Практическая реализация данных принципов в структурах различных процессоров имеет ряд существенных особенностей, связанных с их микроархитектурой. **Микроархитектура** процессора определяет реализацию его внутренней структуры, принципы выполнения поступающих команд, способы размещения и обработки данных.

На данный момент на рынке процессоров лидерами являются 2 компании с разными стратегиями развития и производства – это **Intel** и **AMD**.

2.11.1 Стратегия развития процессоров Intel

Стратегия развития Intel заключается во внедрении новых микроархитектур процессоров, основанных на новых поколениях полупроводниковой производственной технологии. В таблице 2.11.1 приведена стратегия производства процессоров Intel за последние 10 лет. Темпы выпуска инновационных микроархитектур и полупроводниковых технологий основаны на принципе, который корпорация Intel называет моделью «TICK-TOCK» («ТИК-ТАК»). Каждый «TICK» обозначает (табл. 2.11.1) новый этап развития полупроводниковых технологий (техпроцесс – 65 нм, 45 нм, 32 нм, 22 нм, 14 нм, 8 нм и т.д.), а каждый «TOCK» – создание новой микроархитектуры (Intel Core, Nehalem, Sandy Bridge, Haswell, Skylake). Переход на новый техпроцесс сопровождается выпуском соответствующих семейств процессоров (Penryn, Westmere, Ivy Bridge, Broadwell).

Этот цикл, как правило, повторяется каждые 2 года. (Однако с 2014 года переход на новый тех. процесс замедлился. С момента выпуска микроархитектуры Broadwell до 2018 года тех. процесс остался 14 нм, при этом выпущено 2 новых микроархитектуры). Новаторская микроархитектура «обкатывается» на текущем производственном процессе, затем переносится на новую производственную технологию. Данная модель развития позволяет осуществлять внедрение единообразной процессорной микроархитектуры во всех сегментах рынка.

Стратегия развития архитектуры и полупроводниковой технологии, реализуемая корпорацией Intel, не только позволяет выпускать новые решения в соответствии с запланированными темпами, но и способствует внедрению инновационных решений в отрасли на уровне платформ, расширяя использование преимуществ высокой производительности и энергоэкономичности.

Таблица 2.11.1. Стратегия развития процессоров Intel за 10-12 лет

Микроархитектура	Процессоры	Ядра/потоки	Сокет (desktop)	Тех. процесс	Выпуск	Цикл	Описание
Intel Core	Intel Core 2, Xeon	2/2	LGA 775, 771	65 нм	2006 г.	TOCK	Новая микроархитектура
Intel Penryn				45 нм	2008 г.	TICK	Семейство процессоров с новым техпроцессом
Intel Nehalem	Intel Core i5	4/4		45 нм	2009 г.	TOCK	Новая микроархитектура
	Intel Core i7	2/4					
	Pentium G	2/2					
	Celeron G	2/2					
	Xeon						
Intel Westmere	Intel Core i3	2/4	LGA 1366, 1156	32 нм	2010 г.	TICK	Семейство процессоров с новым техпроцессом
	Intel Core i5	2/4					
	Intel Core i7	4/8					
	Pentium G	2/2					
	Celeron G	2/2					
	Xeon						
Intel Sandy Bridge	Intel Core i3	2/4	LGA 2011, 1155	32 нм	2011 г.	TOCK	Новая микроархитектура
	Intel Core i5	4/4					
	Intel Core i7	4/8					
	Pentium G	2/2					
	Celeron G	2/2					
	Xeon						
Intel Ivy Bridge	Intel Core i3	2/4	LGA 2011, 1155	22 нм	2012 г.	TICK	Семейство процессоров с новым техпроцессом
	Intel Core i5	4/4					
	Intel Core i7	4/8					
	Pentium G	2/2					
	Celeron G	2/2					
	Xeon						
Intel Haswell	Intel Core i3	2/4	LGA 2011-3,	22 нм	2013 г.	TOCK	Новая микроархитектура

Микроархитектура	Процессоры	Ядра/потоки	Сокет (desktop)	Тех. процесс	Выпуск	Цикл	Описание
	Intel Core i5	4/4	1150				
	Intel Core i7	4/8					
	Pentium G	2/2					
	Celeron G	2/2					
	Xeon						
Intel Broadwell	Intel Core i3	2/4	LGA 2011-3, 1150	14 нм	2014 г.	TICK	Семейство процессоров с новым техпроцессом
	Intel Core i5	4/4					
	Intel Core i7	4/8					
	Pentium G	2/2					
	Celeron G	2/2					
	Xeon						
Intel Skylake	Intel Core i3	2/4	LGA 1151		2015 г.		
	Intel Core i5	4/4					
	Intel Core i7	4/8					
	Pentium G	2/2					
	Celeron G	2/2					
	Xeon						
Intel Skylake-X	Intel Core i5	4/4	LGA 2066		2017 г.		
	Intel Core i7	4/8,6/12,8/16					
	Intel Core i9	10/20,12/24, 14/18,16/32, 18/36					
Intel Kaby Lake	Intel Core i3	2/4	LGA 1151		2017 г.		
	Intel Core i5	4/4					
	Intel Core i7	4/8					
	Pentium G	2/2					

Микроархитектура	Процессоры	Ядра/потоки	Сокет (desktop)	Тех. процесс	Выпуск	Цикл	Описание
	Celeron G	2/2					
	Xeon						
Intel Coffee Lake	Intel Core i3	2/4	LGA 1151-v2		2017 г.		
	Intel Core i5	6/6					
	Intel Core i7	6/12					
	Pentium G	2/2					
	Celeron G	2/2					
	Xeon						
Intel Coffee Lake-Refresh	Intel Core i3	4/4	LGA 1151-v2	14 нм	2017 г.		
	Intel Core i5	6/6					
	Intel Core i7	8/8					
	Intel Core i9	8/16					
Intel Cannonlake	-	-	-	10 нм	2018 г.	TICK	Семейство процессоров с новым техпроцессом
Intel Icelake	-	-	-	10 нм	2019 г.	TOCK	Новая микроархитектура
Intel Tigerlake	-	-	-		2020 г.		Новая микроархитектура

2.11.2 Стратегия развития процессоров AMD

Изначально, компанией, которая выпускала процессоры для компьютеров, была Intel. Однако правительству США не нравилось, что такая важная для оборонной промышленности и экономики страны деталь выпускается только одной компанией. С другой стороны, были и другие желающие выпускать процессоры.

Таким образом, была основана компания AMD (Advanced Micro Devices). Intel поделилась с ними всеми своими наработками и разрешила AMD использовать свою архитектуру для выпуска процессоров. Но продлилось это недолго, т.к. спустя несколько лет Intel перестала делиться новыми наработками и AMD пришлось улучшать свои процессоры самим.

До 1995 года AMD выпускали процессоры на основе технологий и наработок Intel. В 1995 году AMD выпустили первый самостоятельный процессор AMD K5 и с тех пор до наших дней являются самостоятельной крупной компанией, выпускающей процессоры (и не только).

Определённой и жесткой стратегии (как у Intel) по производству процессоров у AMD нет. В таблице 2.11.2 приведена хронология развития процессоров AMD за последние 10 лет.

Таблица 2.11.2. Стратегия производства процессоров AMD за 10 лет

Микроархитектура	Процессоры	Ядра/потоки	Сокет (desktop)	Тех. процесс	Выпуск
AMD K10	AMD Phenom	4/4	AM2+	65	2008
	AMD Phenom II X4	4/4	AM2+/ AM3	45	2009
	AMD Athlon II	4/4	AM2+/ AM3	45	2009
AMD Bulldozer	AMD FX 4xxx	4/4	AM3+	32	2011
	AMD FX 6xxx	6/6	AM3+		
	AMD FX 8xxx	8/8	AM3+		
AMD Piledriver	AMD FX 4xxx	4/4	AM3+	32	2012
	AMD FX 6xxx	6/6	AM3+		
	AMD FX 8xxx	8/8	AM3+		
	AMD A4	2/2	FM2		
	AMD A6	2/2	FM2		
	AMD A8	4/4	FM2		
	AMD A10	4/4	FM2		
AMD Steamroller	AMD FX 4xxx	2/2	FM2+	28	2014
	AMD FX 6xxx	4/4	FM2+		
	AMD FX 8xxx	8/8	FM2+		
	AMD A4	2/2	FM2+		
	AMD A6	2/2	FM2+		
	AMD A8	4/4	FM2+		
	AMD A10	4/4	FM2+		
AMD Excavator	AMD A6	2/2	AM4	28	2015
	AMD A8	4/4	AM4		
	AMD A10	4/4	AM4		
	AMD A12	4/4	AM4		
AMD Zen	AMD Ryzen 3	4/4	AM4	14	2016
	AMD Ryzen 5	4/8,6/12	AM4		
	AMD Ryzen 7	8/16	AM4		
	AMD Threadripper	12/24,16/32	TR4		
AMD Zen+	AMD Ryzen 5	6/12	AM4	12	2018
	AMD Ryzen 7	8/16	AM4		
	AMD Ryzen Threadripper	12/24,16/32, 24/48,32/64	TR4		
AMD Zen 2	-	-	-	7	2018

2.11.3 Структурная организация процессоров Intel микроархитектур Skylake, Kaby Lake, Coffe Lake

Шестое поколение многоядерных процессоров Intel Core с рабочим названием **Skylake** с полным на то правом можно назвать одним из наиболее масштабируемых и революционных за всю историю архитектуры Core. В этом заявлении нет ни малейшего преувеличения. Так, масштабируемость подтверждает ассортимент из почти 50 наименований Xeon, Core i3/5/7, Core M3/5/7, Pentium и Celeron с впечатляющим разбросом характеристик: от крохотных (20 x 16,5 мм) чипов в компактной корпусировке BGA1515 с TDP 4,5 Вт до мощных разблокированных десктопных LGA1151 процессоров вроде Core i7-6700K с габаритами 37,5 x 37,5 мм и TDP порядка 91 Вт. То есть, 20-кратная масштабируемость по энергопотреблению и 4-кратная по размерам чипа.
Что касается тезиса о революционности, он подтверждается действительно существенным изменением схемотехники и производительности большинства ключевых элементов архитектуры, таких как DDR4/DDR3L, eDRAM, графика HD Intel Graphics 5xx, Iris/Iris Pro и многое другое, чему, соответственно, и посвящён этот материал.

Архитектура процессоров Skylake получила сотни структурных изменений и улучшений, позволивших повысить производительность при снижении потребления энергии.

Вычислительные процессорные ядра при этом, хоть и изменились, но остались примерно сравнимыми по структурному строению с предыдущими поколениями Broadwell и Haswell. Чего нельзя сказать о структурном уровне чипа в целом (рис. 2.11.1), где появился ряд совершенно новых модулей и блоков. Благодаря этому новые чипы Intel Core шестого поколения окончательно превратились из классических процессоров в так называемые "системы на чипе" (SoC, System on Chip).

Стандартный набор компонентов процессора Skylake состоит из двух или четырёх вычислительных ядер (CPU), графической подсистемы, общей кольцевой коммуникационной шины, блока платформенных контроллеров PCH (Platform Controller Hub, порой до сих пор называемого "южным мостом") на многоканальной шине DMI/OPI, интегрированного "расширителя кэша" eDRAM (бывший Crystalwell, опциональный у Haswell), шины PCI Express x16, а также встроенного модуля системных блоков System Agent. В свою очередь, в состав System Agent входят как уже привычные (но значительно переработанные), так и совершенно новые блоки, включая доработанный управляющий блок PCU

(Package Control Unit) с блоком контроля температуры (PECI, Platform Environment Control Interface) и напряжения (SVID, Serial Voltage Identification), контроллер памяти DDR3L/DDR4, блоки мультимедийной обработки и вывода видео, плюс совершенно новый процессор обработки изображений ISP (Image Signal Processing).

Основным усовершенствованием процессорных x86-ядер Skylake, позволяющим говорить о повышении качества предсказания ветвлений, загрузки исполнительного конвейера и, как следствие, более частого одновременного декодирования CISC-инструкций и исполнения до шести микроинструкций за каждый такт, стоит назвать значительно расширенные по сравнению с предыдущими поколениями внутренние буферы.

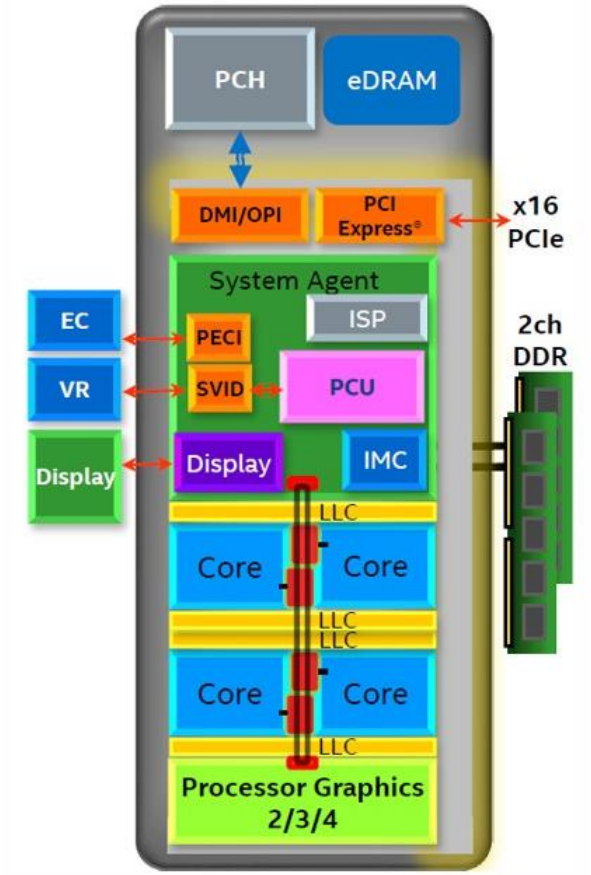


Рис. 2.11.1. Стандартный набор компонентов процессора

На представленном ниже скриншоте из презентации Intel наглядно виден последовательный рост емкости буферов на примере трёх последних поколений архитектуры Core (рис. 2.11.2).

	Sandy Bridge	Haswell	SkyLake
Out-of-order Window	168	192	224 ↑
In-flight Loads	64	72	72
In-flight Stores	36	42	56 ↑
Scheduler Entries	54	60	97 ↑
Integer Register File	160	168	180 ↑
FP Register File	144	168	168
Allocation Queue	28/thread	56	64/thread ↑

Рис. 2.11.2. Последовательный рост ёмкости внутренних буферов процессора

Обработка входящих команд улучшена благодаря ускоренной работе более ёмкого блока предсказания ветвлений, более глубокая буферизация внеочередного исполнения инструкций позволила говорить о более качественном распараллеливании обрабатываемого кода, уменьшении латентности и снижении энергопотребления в моменты простоя (рис. 2.11.3).

В целом архитектура Skylake обладает более глубокой буферизацией данных, чтения/записи, отложенной (write-back) записи, ускоренной производительностью обработки промахов страниц и кэш-памяти L2. Со времён Sandy Bridge почти в полтора раза (до 224) увеличилось окно внеочередного исполнения инструкций, улучшена работа Hyper-Threading за счёт почти удвоенного до 64 на поток окна очереди распределения и сниженного простоя конвейера за счёт оптимизированных алгоритмов улучшенного блока предсказания ветвления. Также появились новые инструкции для лучшего управления загрузкой кэш-памяти, а скорость работы протокола AES криптографического шифрования в GCM- и CBC-режимах выросла на 17% и 33%, соответственно.

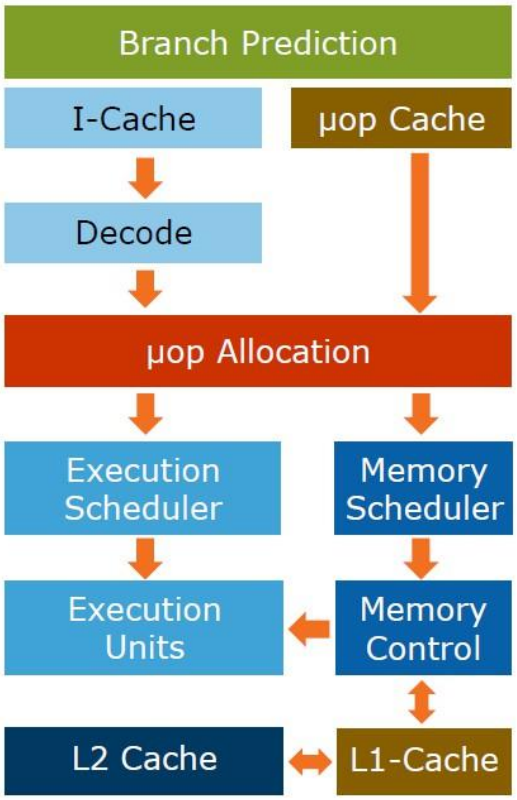


Рис. 2.11.3. Структура ядра процессора

В архитектуре Skylake появилась поддержка наборов новых инструкций MIC и SGX.

Серьёзным изменениям в архитектуре Skylake подверглась кольцевая шина, кэш-память и структура цепей работы с памятью. Согласно данным Intel, пропускная способность кольцевой шины, обеспечивающей обмен между вычислительными ядрами, графической подсистемой, системным агентом, контроллером памяти и кэш-памятью L3 была удвоена по сравнению с поколением Haswell, при этом число используемых для этого транзисторов увеличилось лишь на 50%, а уровень энергопотребления для многих режимов остался на прежнем уровне.

Удвоенная производительность кольцевой шины позволила удвоить скорость работы кэш-памяти L3 при обработке промахов, что вместе с появлением поддержки DDR4 и особенностями работы eDRAM даёт надежду на значительный прирост производительности в некоторых приложениях. Помимо этого улучшенная работа с памятью также сказалась на обеспечении стабильной работы процессора обработки изображений (ISP) и поддержке видеовыхода на три дисплея с разрешением до 4K.

В архитектуре Skylake реализована новая, полностью когерентная структура встроенной DRAM (eDRAM), или Memory Side Cache, способная кэшировать любые данные, включая варианты "некэшируемой памяти", без необходимости очистки для поддержания когерентности, и доступной для использования устройствами ввода-вывода и формирования выходного видеосигнала. Помимо этого графическая подсистема для достижения оптимальной производительности может выбрать режим кэширования определённых данных только в eDRAM без использования кэш-памяти L3.

В отличие от предыдущей архитектуры, где примерно четверть кэш-памяти L3 использовалась для доступа к eDRAM, и при этом eDRAM не имела возможности прямого взаимодействия с остальной системой, в архитектуре Skylake контроллер eDRAM переместился в модуль системного агента, освободив таким образом порядка 512 Кбайт ёмкости кэша L3 и одновременно с этим облегчив доступ другим компонентам ядра к данным в eDRAM (рис. 2.11.4). Отныне Memory Side Cache может взаимодействовать с основной системной памятью напрямую, обеспечивая таким образом обновление экрана без необходимости вывода остальных компонентов процессора из ждущего режима.

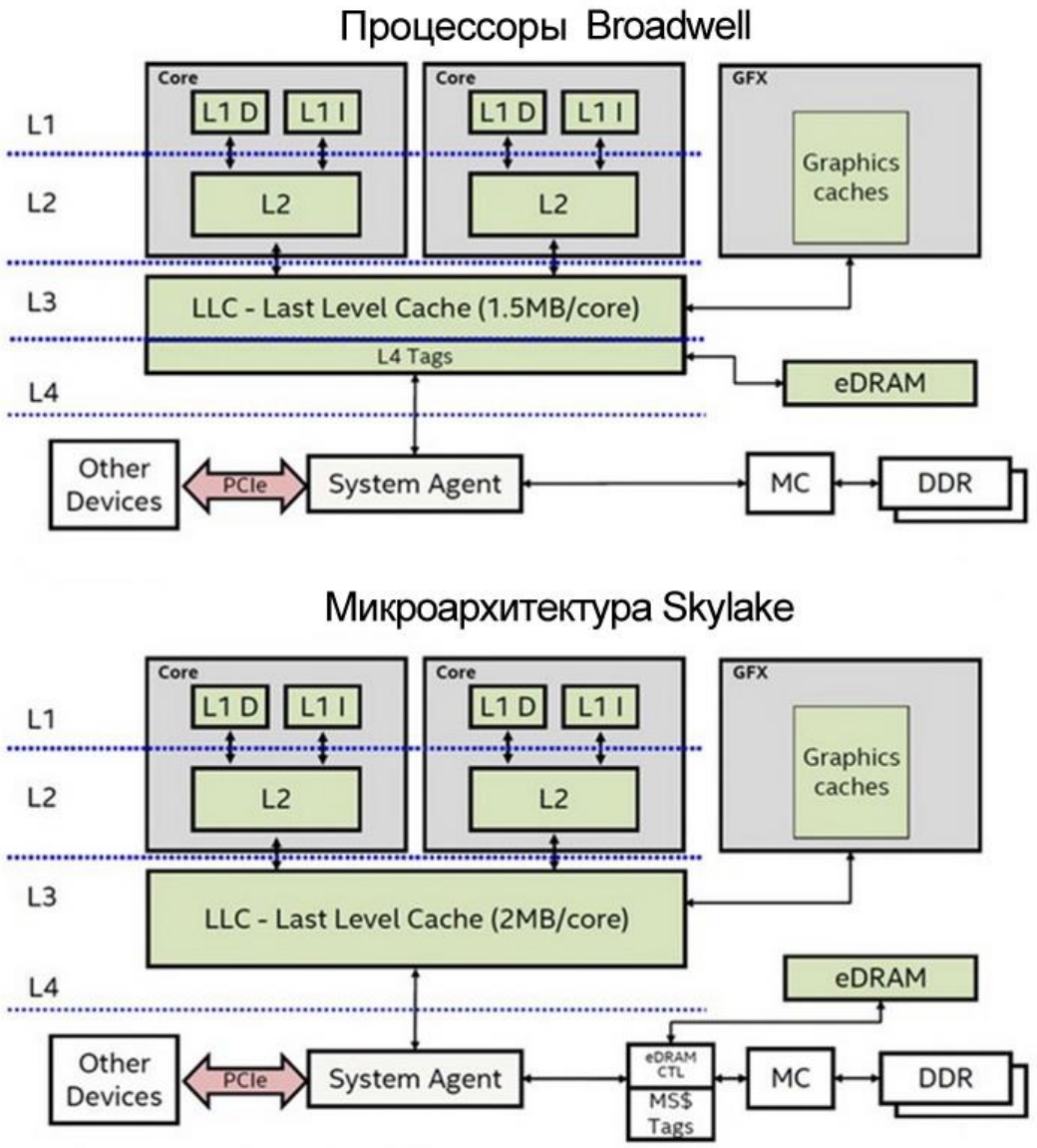


Рис. 2.11.4. Конфигурация процессора с eDRAM

К сожалению, как и в случае с архитектурой Haswell, в Intel так и не озвучили планов использования eDRAM в составе обычных LGA-процессоров на базе Skylake для настольных систем. Не исключено, что на этой стадии развития процессорной микроархитектуры Intel плюсы Memory Side Cache будут реализованы только в чипах для мобильных, встраиваемых и компактных систем.

Полупроводниковая логика и программная прошивка управляющего блока PCU в архитектуре Skylake подверглась значительным доработкам для достижения более динамичных режимов с высокими уровнями производительности и агрессивными алгоритмами энергосбережения, в том числе, в плане сбора внутренней статистики, внешней и

внутренней телеметрии (iMon, Psys) и более тесного взаимодействия со старшими в иерархии системами управления (OS, BIOS).

Самое заметное и, пожалуй, существенное изменение в архитектуре Skylake связано с новым интегрированным контроллером оперативной памяти: он по-прежнему 2-канальный, однако теперь поддерживает динамическую память как предыдущего стандарта DDR3L, так и нового поколения DDR4. Прежде полноценный контроллер DDR4 был на вооружении только серверных чипов Xeon и выполненных на основе их дизайна топовых геймерских LGA 2011 чипах Intel Core i7, но фактически именно начиная со Skylake память DDR4 начинает своё широкомасштабное наступление на рынок настольных и мобильных систем.

Модуль аппаратной обработки мультимедийного контента и финального рендеринга видеопотока в архитектуре Skylake также значительно доработан и улучшен. В частности, аппаратный кодек HEVC/H265 нового поколения обладает повышенной производительностью с одновременным уменьшением энергопотребления; есть отдельный аппаратный кодек AVC с очень низким энергопотреблением; обеспечивается вывод изображения на три независимых 4K-дисплея со сниженным на 40%-60% потреблением энергии даже в самом тяжёлом режиме воспроизведения 4K-видео.

В дополнение, архитектура Skylake также поддерживает инициативу Intel по отказу от проводных подключений для беспроводной передачи мультимедийного контента с помощью технологий Intel WiDi или Pro WiDi с компьютеров на телевизоры, мониторы или проекторы.

Впервые в составе архитектуры для массовых процессоров Skylake (а не специализированных SoC) появился так называемый встроенный процессор обработки изображений – ISP (Image Signal Processing), что особенно актуально для смартфонов, планшетов и ноутбуков. В частности, ISP обладает встроенным интерфейсом CSI (Camera Sensor Interface) с поддержкой до четырех внешних цифровых камер/сенсоров с разрешением до 13 Мп, правда, с одновременным обслуживанием только двух из них. Аппаратная обвязка CSI поддерживает расширенный список технологий для полноценной обработки фото и видео, включая распознавание и запоминание лиц, групповые снимки, многопоточный захват, съёмку с расширенным динамическим диапазоном (HDR), съёмку при слабом освещении, серийную съёмку и многое другое.

Роль графических ядер, встроенных в процессоры, с каждым годом увеличивается. И это связано не столько с ростом их 3D-производительности, столько с тем, что встроенные GPU берут на себя

всё новые функции, такие как параллельные вычисления или кодирование и декодирование мультимедийного контента. Исключением не стало и графическое ядро Skylake. Intel относит его к следующему, девятому поколению, и это значит, что в нём таится немало сюрпризов. Однако начать стоит с того, что GPU, реализованный в Skylake, как и его предшественники, сохранил традиционный модульный дизайн. Таким образом, мы вновь имеем дело с целым семейством решений разного класса: на базе имеющихся строительных блоков нового поколения Intel может собирать кардинально различающиеся по уровню производительности GPU. Подобная масштабируемость сама по себе новинкой не является, но в Skylake возросла не только максимальная производительность, но и число доступных вариантов графического ядра.

Итак, графическое ядро Skylake может быть построено на базе одного или нескольких модулей, каждый из которых обычно включает в себя по три секции. Секции объединяют по восемь исполнительных устройств, на которые ложится основная часть обработки графических данных, а также содержат базовые блоки для работы с памятью и текстурные семплеры. Помимо исполнительных устройств, сгруппированных в модули, графическое ядро содержит и внемодульную часть, отвечающую за фиксированные геометрические преобразования и отдельные мультимедийные функции.

На самом верхнем уровне иерархии графическое ядро Skylake очень похоже на ядро, реализованное в Broadwell. Однако если углубиться в подробности, то нетрудно найти и заметные изменения.

Во-первых, внемодульная часть вынесена теперь в отдельный энергетический домен, что позволяет задавать ей частоту и отправлять её в сон отдельно от исполнительных устройств. Это значит, что, например, при работе с технологией Quick Sync, которая реализуется как раз силами внемодульных блоков, основная часть GPU может быть отключена от линий питания в целях снижения энергопотребления. Кроме того, независимое управление частотой внемодульной части позволяет лучше подстраивать её производительность под конкретные нужды модулей графического ядра.

Во-вторых, в то время как графическое ядро Broadwell могло основываться лишь на одном или двух модулях, получая в своё распоряжение 24 или 48 исполнительных устройств (для энергоэффективных и бюджетных процессоров мог использоваться один модуль с отключенными секциями, что давало меньшее, чем 24, число исполнительных устройств), в Skylake может применяться от одного до трёх модулей.

Благодаря этому в дополнение к привычным конфигурациям GT1/GT2/GT3 в семействе процессоров Skylake будет доступно ещё более мощное ядро GT4 (рис. 2.11.5), которое получит 72 исполнительных устройства.

Однако пиковая производительность самих исполнительных устройств в Skylake не изменилась – каждое такое устройство может выполнять до 16 32-битных операций за такт. При этом оно способно исполнять 7 вычислительных потоков одновременно и имеет 128 32-байтовых регистров общего назначения.

В-третьих, варианты ядра GT3 и GT4 могут быть дополнительно усилены eDRAM-буфером объёмом 64 или 128 Мбайт соответственно, что даёт модификации GT3e и GT4e. Процессоры Broadwell комплектовались лишь одним вариантом eDRAM – объёмом 128 Мбайт. В Skylake же этот дополнительный буфер не только изменил алгоритм работы, став «кешем на стороне памяти», но и приобрёл некоторую гибкость конфигурации. Однако его исполнение останется старым – он будет представлен отдельным 22-нм кристаллом, монтируемым на процессорную плату по соседству с основным чипом.

Полная структурная схема ядра Skylake приведена на рис. 2.11.6.

Процессоры поколения **Kaby Lake**, которые появились на рынке позже, стали первым и очень ярким примером попыток Intel продать клиентам тот же Skylake во второй раз. Близкие родственные связи между двумя поколениями процессоров особо и не скрывались. Intel честно говорила, что Kaby Lake – это уже не «тик» и не «так», а простая оптимизация предыдущего дизайна. При этом под словом «оптимизация» понимались некие улучшения в структуре 14-нм транзисторов, которые открывали возможность увеличения тактовых частот без изменения рамок теплового пакета. Для видоизменённого техпроцесса был даже придуман специальный термин «14+ нм». Благодаря этой производственной технологии старший массовый десктопный процессор Kaby Lake, получивший наименование Core i7-7700K, смог предложить пользователям номинальную частоту 4,2 ГГц и частоту турборежима 4,5 ГГц.

В виду того, что Intel не справились с вводом в строй 10-нм технологии, на рынок была выведена ещё одна разновидность процессоров, построенных на всё той же микроархитектуре Skylake – **Coffee Lake**. Процессоры Coffee Lake получили важное структурное отличие от своих предшественников: число ядер в них было увеличено до шести штук, что с массовой платформой Intel произошло впервые. Однако при этом никаких изменений на **уровне микроархитектуры** вновь введено не

было: Coffee Lake по сути – шестиядерный Skylake, собранный на основе точно таких же по внутреннему устройству вычислительных ядер, которые снабжены увеличенным до 12 Мбайт L3-кешем (по стандартному принципу 2 Мбайт на ядро) и объединены привычной кольцевой шиной.

Рис. 2.11.5 Структура графического ядра Skylake

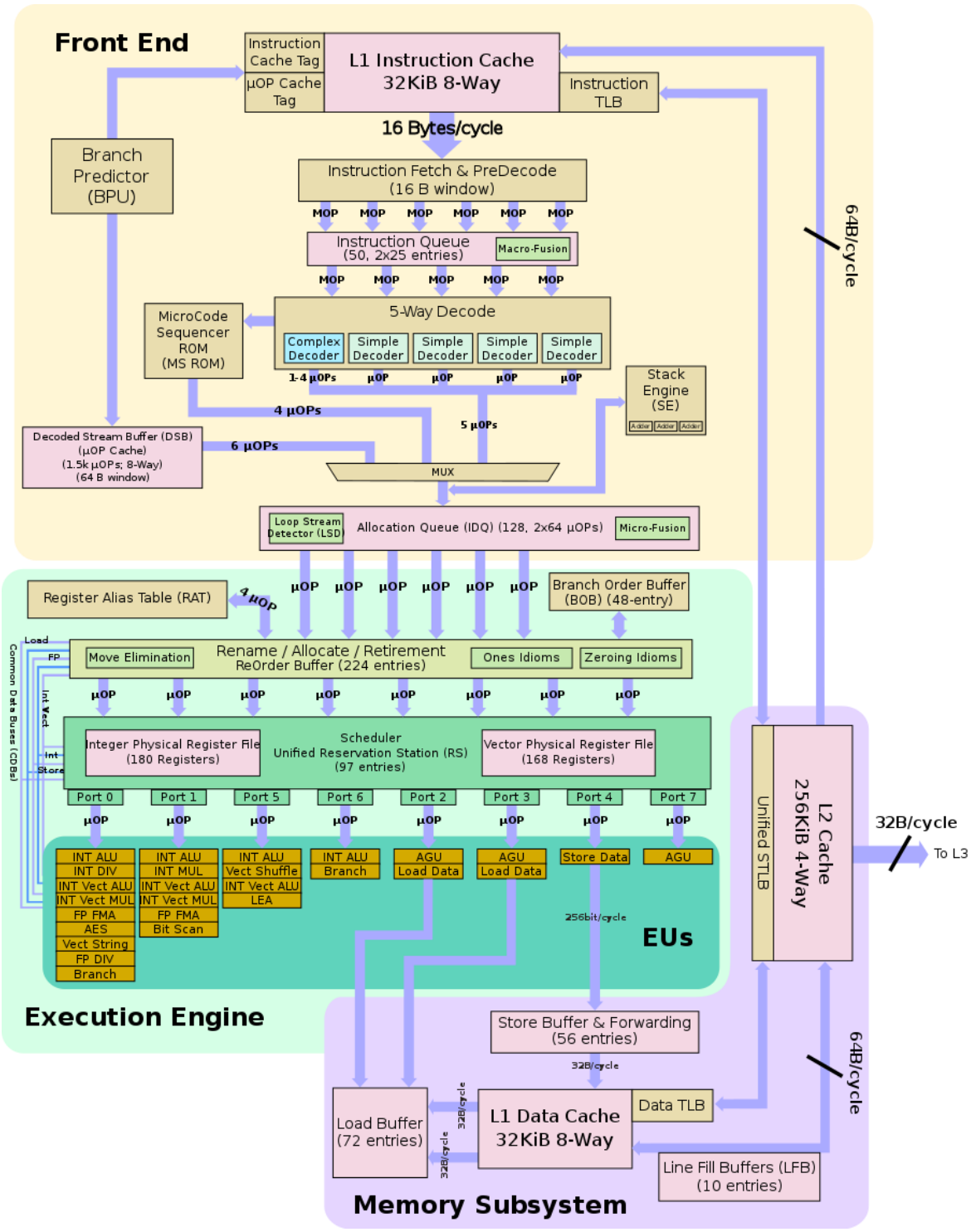


Рис. 2.11.6. Структура ядра Skylake

2.11.4 Структурная организация процессоров AMD микроархитектуры Zen

Архитектура Zen — это принципиально новая конструкция процессоров x86. Она дает начало новому поколению высокопроизводительных вычислительных устройств от компании AMD, которые появились в 2017 году и будут разрабатываться в дальнейшем.

На основе этой микроархитектуры вышли процессоры AMD под торговыми марками Ryzen и EPYC. Чипы на этой микроархитектуре делятся на три группы: две группы торговой марки Ryzen — Summit Ridge (настольные процессоры без графических ядер) и Raven Ridge (настольные и мобильные процессоры со встроенными графическими ядрами) и одну группу торговой марки EPYC — Naples (серверные процессоры).

По словам AMD, основное внимание уделялось увеличению количества операций за такт (ipc, instructions per clock). Переход от микроархитектуры модулей, используемой в bulldozer, к полноценным ядрам, как ожидалось, поможет увеличить производительность на ядро в операциях с плавающей точкой за счёт большего количества блоков fpu.
Особенности микроархитектуры:

- два потока на ядро;
- снижение числа ошибок прогнозирования;
- кэш декодированных микроопераций;
- увеличенный объём кэш-памяти и скорости доступа к ней (7 циклов вместо 9);
- по 8 МБ общей кэш-памяти третьего уровня типа victim на каждый комплекс из 4 ядер,
- по 512 КБ индивидуальной в 2 раза более быстрой кэш-памяти второго уровня на каждое ядро (включает кэш первого уровня),[9]
- по 64 КБ на инструкции и 32 КБ на данные в индивидуальной в два раза более быстрой кэш-памяти первого уровня на каждое ядро (включён в кэш второго уровня);
- два блока с реализацией аппаратных ускорителей стандарта шифрования AES.

В отличие от предыдущих архитектур AMD отошла от концепции модулей CPU. Таким образом, имеется вариант многоядерного дизайна с монолитным блоком ядер. Диспетчеры декодирования и целых чисел больше не работают в двух модулях, они выполнены в виде единого блока (рис. 2.11.7). Для целочисленных вычислений доступно уже не

восемь, а только шесть конвейеров. Вычисления с плавающей запятой выполняются на двух блоках FMAC (совмещенное умножение-сложение, fused multiply–accumulate) уже не со 128-битной шириной, а с 256-битной.

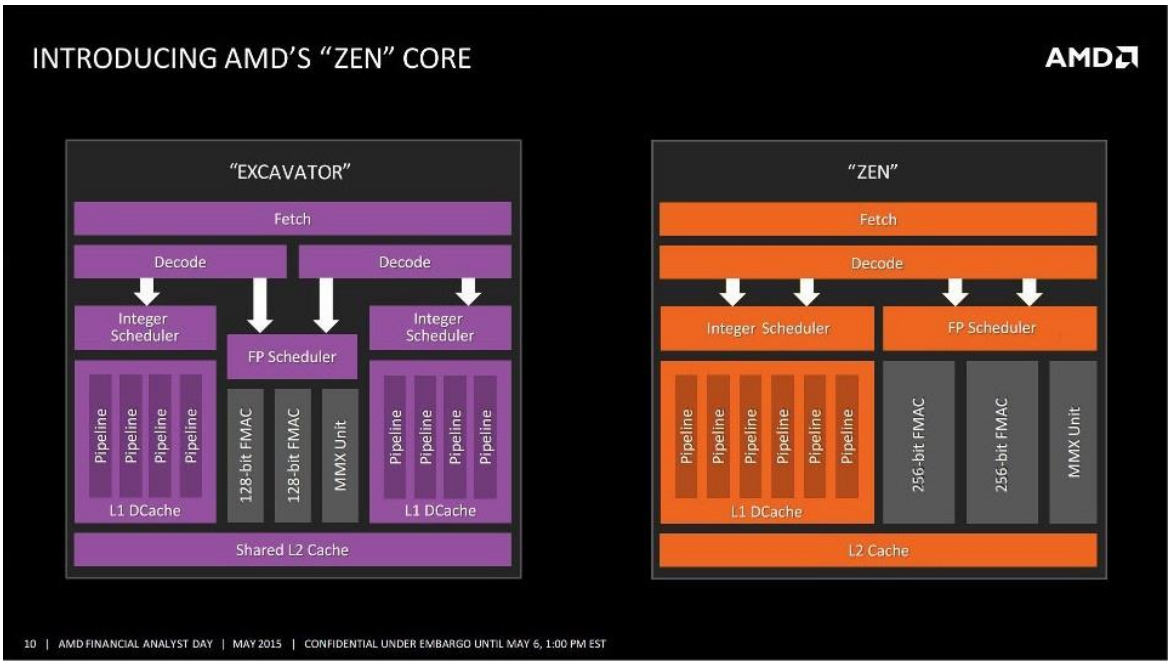


Рис. 2.11.7 Диаграмма архитектуры Zen

Операции MAC подразумевают умножение двух чисел, результат складывается с аккумулятором. Операции MAC могут улучшить точность результата, поскольку округление выполняется только в конце операции MAC, промежуточные результаты просчитываются с полной точностью без округления. Новые блоки FMAC в два раза шире, что увеличивает их аппаратную сложность. Однако такие блоки позволяют дать ещё более высокую точность, что актуально для сложных расчётов.

Кэш L2 имеет ёмкость всего 512 кбайт на ядро, хотя у Excavator объём мог достигать 2 Мбайт. Поскольку новые ядра могут высчитывать данные быстрее, то и более крупный кэш больше не нужен. Примерно такой же подход Intel использовала в Haswell. Скорее всего, AMD будет использовать в архитектуре Zen крупный кэш L3 или L4, где данные будут накапливаться, а также ядра смогут обмениваться данными. Там же будет обеспечиваться доступ к оперативной памяти и интерфейсу PCI Express.

На рис. 2.11.8. приведена структура полного процессора с ядрами Zen. Ядра Zen входят в комплексы, называемые CCX. В каждом CCX обычно расположены 4 ядра Zen с общим для всех ядер кэшем третьего

уровня, объёмом 8 МБ на комплекс. Кэш третьего уровня по большей части эксклюзивный, в то время как данные кэша первого уровня обязательно присутствуют в кэше второго уровня . Каждое ядро в комплексе может обратиться к ячейкам кэша любого уровня примерно с одной и той же скоростью, однако в рамках CCX имеется некоторое замедление при обращении к дальней 4МБ половине L3 кэша, а доступ к 8 МБ L3 памяти в соседний CCX проходит с почти на порядок более низкой скоростью. На рис. 2.11.9 приведена полная структура ядра Zen.

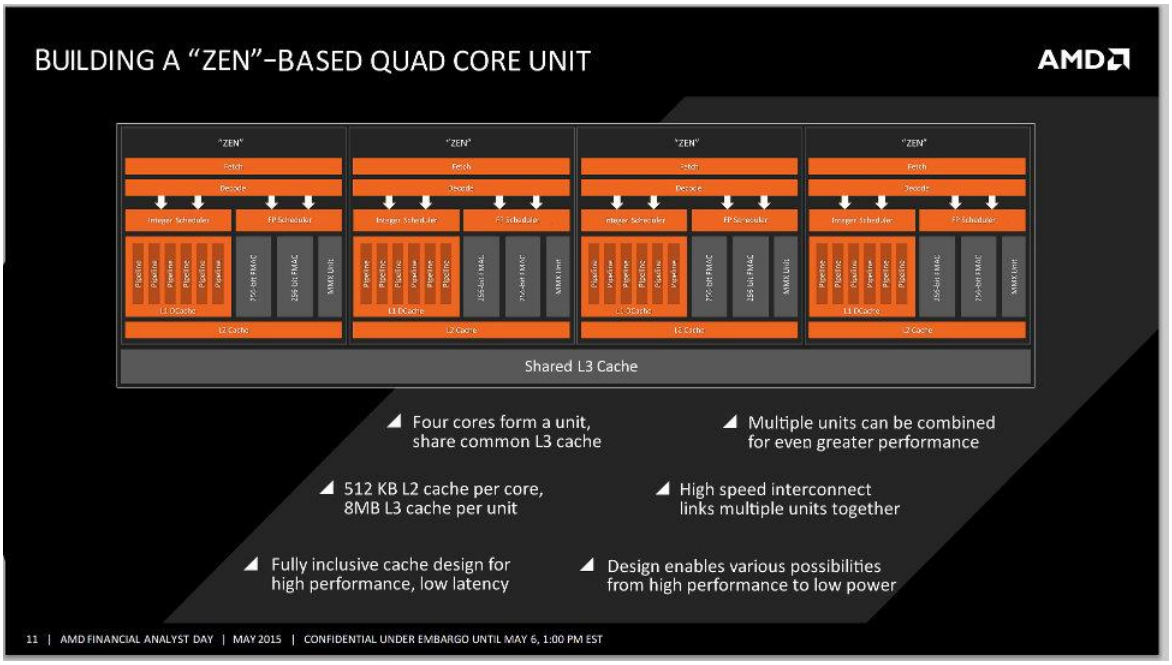


Рис. 2.11.8. Блок из 4-х ядер Zen

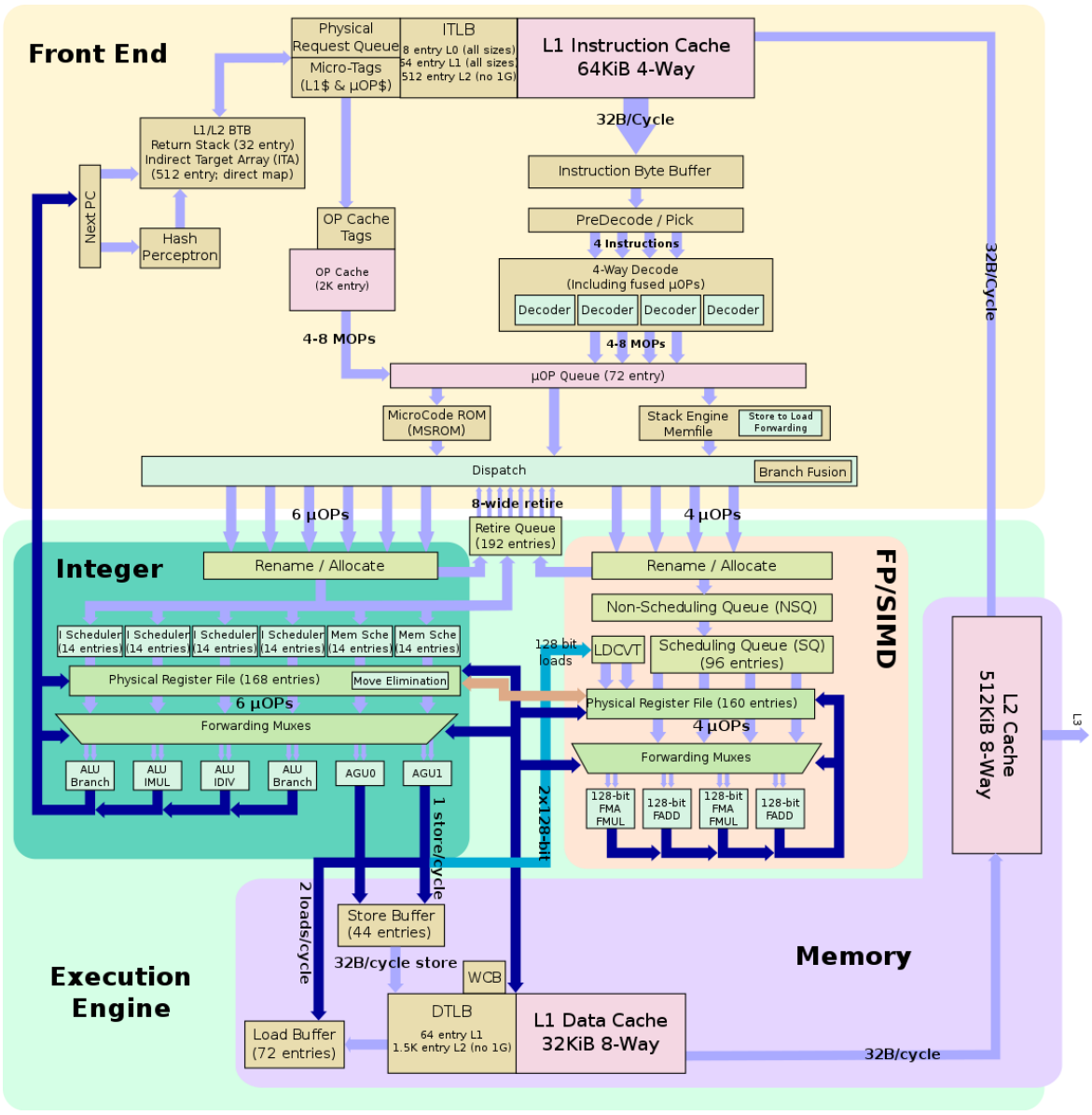


Рис. 2.11.9. Структура ядра Zen

2.11.5 Аппаратные уязвимости современных микропроцессоров

Несмотря на совершенствование микроархитектур современные процессоры не лишены ряда аппаратных уязвимостей, снижающих производительность. Наиболее популярными из современных уязвимостей являются **Spectre** и **Meltdown**.

Spectre — аппаратная уязвимость, ошибка в большинстве современных процессоров, имеющих спекулятивное выполнение команд и развитое предсказание ветвлений, позволяющая проводить чтение данных через сторонний канал в виде общей иерархии кэш-памяти. Затрагивает большинство современных микропроцессоров, в частности, ар-

хитектур x86/x86-64 (Intel и AMD) и некоторые процессорные ядра ARM. Уязвимость потенциально позволяет локальным приложениям (локальному атакующему, при запуске специальной программы) получить доступ к содержимому виртуальной памяти текущего приложения или других программ.

Meltdown — аппаратная уязвимость, обнаруженная в ряде микропроцессоров, в частности, производства Intel и архитектуры ARM. Meltdown использует ошибку реализации спекулятивного выполнения команд в процессорах Intel и ARM, из-за которой при спекулятивном выполнении инструкций чтения из памяти процессор игнорирует права доступа к страницам. Уязвимость позволяет локальному атакующему (при запуске специальной программы) получить несанкционированный доступ на чтение к привилегированной памяти (памяти, используемой ядром операционной системы. Уязвимость не затрагивает микропроцессоры корпорации **AMD**.

В целом, Meltdown и Spectre — это разновидность атаки по времени. Основная разница между ними состоит в том, как они вызывают кеширование данных из защищенных страниц памяти. Meltdown пользуется задержкой в обработке исключения, а Spectre обманывает блоки предсказания.

Сама идея внеочередного исполнения инструкций была предложена IBM в 1963 году. Рядовые пользователи столкнулись с ней только в 1995 году, когда появился первый Pentium Pro. С тех пор все современные процессоры используют архитектуру с внеочередным (out-of-order) и одновременно упреждающим (или спекулятивным — speculative) механизмом исполнения команд.

Современное процессорное ядро для исполнения инструкций использует многоуровневый конвейер с нелинейными путями. Все входящие инструкции в нем кешируются и декодируются по нескольку штук за такт. Они дробятся на микрооперации, которые переупорядочиваются в собственном буфере (ROB, Re-Order Buffer). Ни одна из них не работает с реальными адресами памяти и физическими регистрами процессора. Блок управления памятью транслирует виртуальные адреса и определяет параметры доступа к ним.

После ROB микрооперации отправляются в станцию резервации, где обрабатываются параллельно на разных исполнительных блоках. Современные процессоры используют разные типы кешей для ускорения обработки данных, изначально загружаемых в не слишком быструю оперативную память. Простейший кеш для инструкций применялся еще в 1982 году, а кеш данных используется с 1985 года. Сейчас они оба от-

носятся к базовому кешу первого уровня (L1), помимо которого появились L2 и L3 общего назначения.

Все эти архитектурные особенности создавались для уменьшения времени простоя отдельных исполнительных блоков процессора. Пока один блок ожидает загрузки данных из оперативной памяти для своей инструкции, другой параллельно обрабатывает следующие. В этом ему помогает MMU (Memory Management Unit) и общая архитектура конвейера, который старается работать на упреждение и постоянно подгружает наиболее востребованные данные в кеш.

Ключевой момент в этой схеме состоит в том, что в большинстве процессоров все проверки легитимности исполнения инструкций и даже контроль за разделением доступа к памяти в самом MMU происходят уже на заключительном этапе обработки инструкций, когда начинается выстраивание промежуточных результатов в порядке исходной очереди команд. Запрещенные операции игнорируются (вызывают исключение), а невостребованные данные при этом остаются в кеше. Собственно, это и есть главная проблема.

В большей мере задержками в проверке условий и длинным забеганием вперед по ветвям инструкций грешат процессоры Intel с длинным конвейером (кроме Itanium и первых Atom), но AMD и даже ARM это тоже касается — просто в для них потребуется более точный таймер, другие размеры массивов и техники забивания кеша. Если Meltdown для них пока не смогли убедительно воспроизвести, то Spectre — вполне.

2.11.6 Микропроцессоры семейства «Эльбрус»

В 1969 году в связи с необходимостью в высокопроизводительной вычислительной технике для оснащения в нашей стране стратегических систем специального назначения возникла идея разработки многопроцессорных вычислительных комплексов (МВК) семейства «Эльбрус». Основоположником идеи был Лебедев С.А., а главным конструктором проекта в Институте точной механики и вычислительной техники (ИТМ и ВТ) стал Бурцев В.С.

Разработка МВК «Эльбрус-1» началась в 1973 г., закончена в 1979 г. и сдана государственной комиссии в 1980 г. Данный вычислительный комплекс был построен на базе TTL-микросхем.

Затем был построен МВК «Эльбрус-2». Годы разработки 1977-1984, сдан в 1985 г. Данный комплекс в течении нескольких лет использовался в центральных объектах стратегических систем страны.

Следующим проектом стал МВК «Эльбрус-3». Руководителем проекта был Бабаян Б.А. Опытный образец данного комплекса был изготовлен в 1990 году, но в связи с прекращением финансирования отладка образца не была завершена.

Стало ясно, что будущие поколения вычислительной техники должны базироваться на микропроцессорах. В 1992 г. коллектив разработчиков машин семейства «Эльбрус» выделился в компанию ЗАО «МЦСТ – Московский центр SPARC-технологий» и совместно с ОАО «ИНЭУМ имени И.С. Брука» начал вести работы над микропроцессорной реализацией разработанной архитектуры.

Основой серии осталась оригинальная архитектура «Эльбрус», а также добавилась открытая архитектура SPARC «Scalable Processor Architecture» (табл. 2.11.3).

Таблица 2.11.3. Разработка линии микропроцессоров «Эльбрус»

Микропроцессор	«Эльбрус»	«Эльбрус-2С»	«Эльбрус-4С»	«Эльбрус-8С»
Технологическое разрешение, нм	130	90	65	28
Год выпуска	2007	2010	2012	2014
Чисто процессорных ядер	1	2	4	8
Частота, МГц	300	600	800	1300
Производительность, GFlops	4,8	19,2	50	250
Мощность, Вт	6	16	25	60

Кроме того, появилось семейство микропроцессоров МЦСТ-R. Это семейство включает в себя микропроцессоры: МЦСТ-R150, МЦСТ-

R500, МЦСТ-R500S, МЦСТ-R1000 и МЦСТ-R1000M. Все микросхемы реализуют архитектуру SPARC.

Среди особенностей архитектуры «Эльбрус» инженеры МЦСТ выделяют следующие:

- 6 каналов арифметико-логических устройств, работающих параллельно;
- регистровый файл из 256 84-разрядных регистров;
- аппаратная поддержка циклов, в том числе с конвейеризацией;
- программируемое асинхронное устройство предварительной подкачки данных с отдельными каналами считывания;
- поддержка спекулятивных вычислений и однобитовых предикатов;
- широкая команда (VLIW-архитектура), способная при максимальном заполнении задать в одном такте до 23 операций (более 33 операций при упаковке операндов в векторные команды).

Кроме того, в архитектуре присутствует режим x86-совместимости. Для этого реализована система двоичной трансляции двоичных кодов x86 в коды процессра «Эльбрус».

Микропроцессор «Эльбрус-8С»

В июне 2014 г. была запущена в производство опытная партия универсальных микропроцессоров «Эльбрус-8С». Данный микропроцессор – это полностью российская разработка. В таблице 2.11.4 приведены основные характеристики микропроцессора «Эльбрус-8С».

Таблица 2.11.4.Технические характеристики

Параметр	Значение	Примечание
Архитектура процессора	«Эльбрус»	Количество вычислительных устройств с плавающей запятой увеличено с 4 до 6
Технологический процесс	28 нм	
Рабочая тактовая частота	1300 МГц	Расчетное значение
Количество ядер	8	
Производительность	250 GFlops	На операциях с одинароной точностью (FP32)
Кэш-память 2-го уровня	8 x 512 Кбайт	Отдельная кэш-память для каждого ядра
Кэш-память 3-го уровня	16 Мбайт	Разделяемая между всеми ядрами
Тип контроллеров памяти	DDR3-1600	С поддержкой ECC
Количество контроллеров памяти	4	
Поддержка многопроцессорных систем	До 4-х процессоров	

Параметр	Значение	Примечание
Каналы межпроцессорного обмена (пропускная способность)	3 (16 Гбайт/с)	Каналы дуплексные (пропускная способность в каждую сторону – 8 Гбайт/с)

Базовой операционной системой для платформы «Эльбрус» является ОС «Эльбрус», построенная на базе ядра Linux. Система программирования платформы поддерживает языки C, C++, Java, Фортран-77, Фортран-90.

На рис. 2.11.10 приведена структура 8-ядерного процессора Эльбрус-8С.

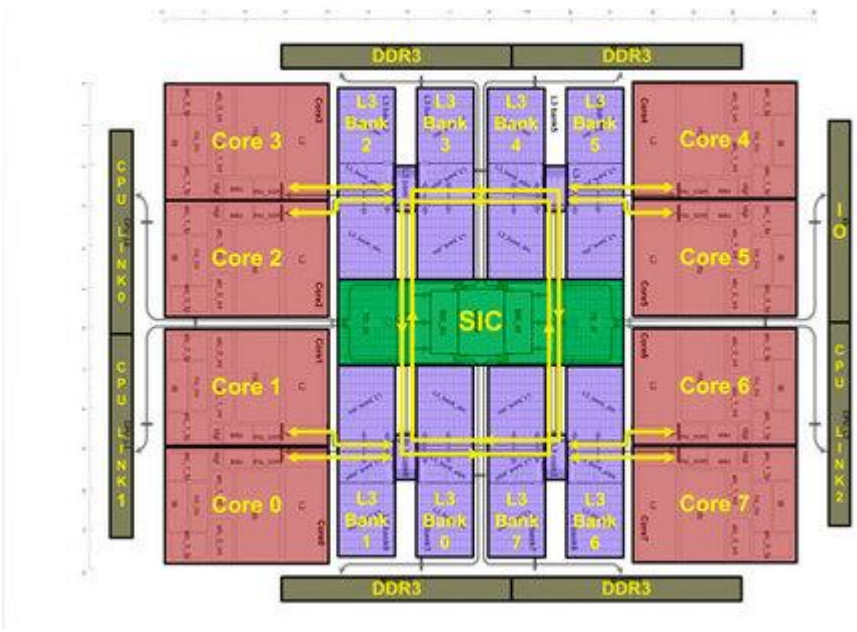


Рис. 2.11.10 Структура Эльбрус-8С

На базе микропроцессора «Эльбрус-8С» планируется организовать массовое производство серверов, рабочих станций и других средств вычислительной техники, предназначенных для применения в государственных учреждениях и бизнес-структурах, предъявляющих повышенные требования к информационной безопасности, а также для применения в области высокопроизводительных вычислений, обработки сигналов, телекоммуникации.

2.12 Системы на кристалле

2.12.1 Общее определение

Прогресс компьютерных технологий всегда был в большей или меньшей степени обусловлен технологическими прорывами в области

электронных систем – созданием транзисторов, интегральных микросхем, достижениями в области технологий передачи данных, появлением и поэтапным совершенствованием микропроцессоров, распространением микроконтроллеров, появлением доступных ПЛИС. К настоящему времени возможности микроэлектроники достигли уровня, когда все элементы сложной вычислительной системы реализуемы в виде одной интегральной микросхемы – «**системы на кристалле**» (**СнК, System on Chip, SoC**) – СБИС (сверхбольшая интегральная схема), интегрирующей на кристалле различные функциональные блоки, которые образуют законченное изделие для автономного применения в электронной аппаратуре. На рис. 2.12.1 приведена общая структурная схема системы на кристалле.

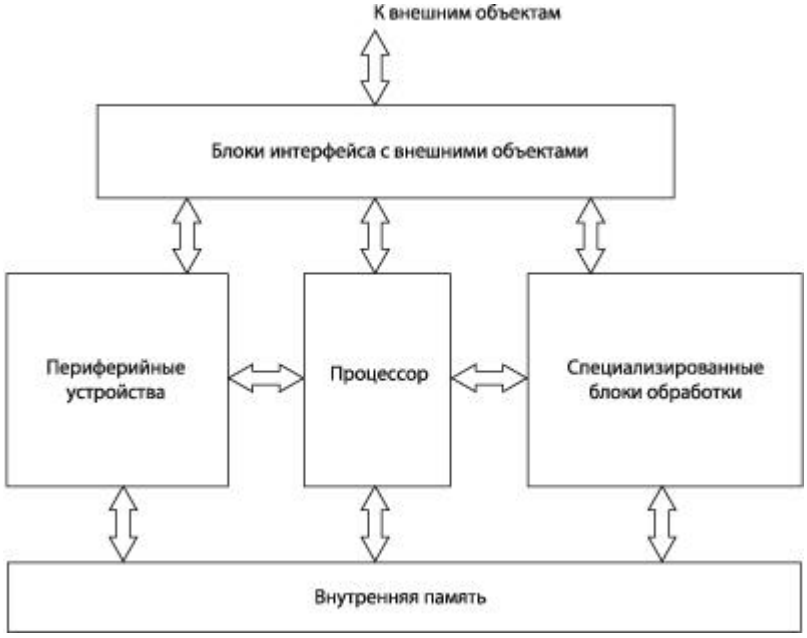


Рис. 2.12.1 Общая структурная схема СнК

В зависимости от назначения она может оперировать как цифровыми сигналами, так и аналоговыми, аналого-цифровыми, а также частотами радиодиапазона. Как правило, применяются в портативных и встраиваемых системах.

Основным цифровым блоком обычно является **процессор**, выполняющий программную обработку цифровых данных. Специализированные блоки обработки обеспечивают аппаратное выполнение функций, специфических для данной системы. Это могут быть, например, блоки цифровой обработки сигналов (DSP), аналоговые схемы, преобразователи потоков данных и др. устройства. Различные типы модулей памяти (SRAM, DRAM, ROM, EEPROM, Flash) могут входить в состав СнК или подключаться к ней как внешние блоки. Таймеры, АЦП и ЦАП, широт-

но-импульсные модуляторы и другие цифровые устройства могут интегрироваться в состав СнК в качестве периферийных устройств. Интерфейс с внешними устройствами обеспечивается с помощью параллельных и последовательных портов, различных шинных и коммуникационных контроллеров и других интерфейсных блоков, в т.ч. аналоговых (усилителей, преобразователей). Состав блоков, интегрируемых в конкретной СнК, варьируется в зависимости от ее функционального назначения. Организация связей между блоками системы также может быть различной: возможно использование различных стандартизованных шин или специализированных локальных интерфейсов.

Как видно из рисунка 2.12.1, структуру СнК составляют в основном те же функциональные блоки, которые входят в состав сложнофункциональных СБИС класса **микроконтроллеров** и **микропроцессоров**. Фактически современные СнК отличаются от микроконтроллеров только наличием специализированных блоков обработки данных. Выпуск микроконтроллеров (называвшихся прежде однокристальными микроЭВМ) начался в 1981 г. Таким образом, можно считать, что СнК без специализированных блоков обработки производятся и применяются уже более 30 лет.

Сейчас, наиболее популярным применением СнК являются мобильные устройства и смартфоны. В современных смартфонах можно встретить СнК семейства **Snapdragon** компании **Qualcomm**, базирующиеся на архитектуре ARM и позиционирующихся как платформа для смартфонов, ноутбуков и планшетов. Также, одним из самых популярных производителей и поставщиков процессоров и СнК для мобильных телефонов является бесфабричная компания **MediaTek**.

Проектирование СнК

В большинстве случаев СнК представляет собой цифровую СБИС, которая может также содержать ряд аналоговых блоков. Поэтому для проектирования СнК используются те же методы и средства, что и для СБИС. Эти средства реализованы в виде систем автоматизированного проектирования (САПР), поставляемых компаниями Cadance, Synopsis, Mentor Graphics и др. В качестве элементной базы эти САПР используют библиотеки функциональных элементов, в состав которых входят как простые логические вентили и триггеры, так и макроэлементы, выполняющие более сложные функции: регистры, счетчики, сумматоры, умножители, арифметико-логические устройства и т. д.

При разработке микроконтроллеров в 90-х гг. прошлого века широкое распространение получила концепция создания микроконтрол-

лерных семейств, имеющих одинаковое процессорное ядро и различающихся набором периферийных устройств и объемом внутренней памяти. Для реализации этой концепции при проектировании СБИС микроконтроллеров кроме функциональных библиотек стали использоваться сложно-функциональные блоки (СФ-блоки) — процессоры, таймеры, АЦП, различные интерфейсные блоки (UART, SPI, CAN, Ethernet и т.д). Эти СФ-блоки формировали верхний уровень функциональных библиотек, используемых разработчиками и производителями микроконтроллеров. Они были достаточно жестко ориентированы на конкретную технологию компании-производителя, являясь внутрифирменной материальной ценностью.

Повышение сложности проектируемых СБИС, жесткие требования к срокам их проектирования (сокращение времени выхода изделия на рынок) поставили перед разработчиками новые проблемы. В сложившихся условиях самостоятельное проектирование разработчиком СнК всех СФ-блоков, входящих в ее состав, не всегда целесообразно. Поэтому в последние годы широкое распространение получила практика разработки отдельных СФ-блоков для их последующего представления на рынок средств проектирования СнК. СФ-блоки, предназначенные для использования в разнообразных проектах, стали называть IP (Intellectual Property) модулями, тем самым подчеркивается, что эта продукция является предметом интеллектуальной собственности.

СФ-блоки, используемые при проектировании СнК, имеют две основные формы представления:

- в виде топологических фрагментов, которые могут быть непосредственно реализованы в физической структуре кристалла — аппаратно реализованные (hard) СФ-блоки;
- в виде моделей на языке описания аппаратуры (Verilog, VHDL), которые средствами САПР могут быть преобразованы в топологические фрагменты для реализации на кристалле — синтезируемые (soft) СФ-блоки.

Таким образом, разработчик может либо непосредственно «вмонтировать» в структуру проектируемой СБИС топологически готовый СФ-блок, либо использовать имеющуюся модель СФ-блока и выполнить его схемотехническое и топологическое проектирование в составе реализуемой СБИС СнК.

В процессе проектирования СнК разработчик имеет возможность выбора следующих решений:

- самостоятельная разработка необходимых СФ-блоков;

- покупка СФ-блоков у ведущих разработчиков и производителей микросхем;
- поиск и применение СФ-блоков, предоставляемых в открытом доступе.

Каждый из этих вариантов имеет свои достоинства и недостатки. Как уже отмечалось, самостоятельная разработка всех СФ-блоков может привести к увеличению сроков проектирования и задержке выпуска конечного изделия. Покупка СФ-блоков сопряжена с определенными финансовыми затратами, повышающими стоимость разработки. Применение СФ-блоков, имеющихся в свободном доступе, возможно только после их тщательной верификации, что требует обычно значительных временных затрат. При выполнении каждого проекта разработчик должен провести оценку поставленных требований и имеющихся ресурсов, чтобы выбрать оптимальный вариант реализации СнК.

Таким образом, основная особенность проектирования СнК — возможность использования достаточно широкой номенклатуры синтезируемых СФ-блоков, имеющихся на рынке и в свободном доступе, которые могут быть реализованы на базе различных функциональных библиотек и технологий и интегрированы в кристалл средствами современных САПР.

2.12.2 Микроконтроллеры

Микроконтроллер – это специальная микросхема, предназначенная для управления различными электронными устройствами. Микроконтроллеры впервые появились в том же году, что и микропроцессоры общего назначения (1971). Микроконтроллер сочетает на одном кристалле функции процессора и периферийных устройств, содержит ОЗУ и (или) ПЗУ и является, по сути, **однокристальным компьютером,** способный выполнять относительно простые задачи. **Отличается** от **микропроцессора** интегрированными в микросхему устройствами ввода-вывода, таймерами и другими периферийными устройствами.

Микроконтроллеры выпускают десятки компаний, причем производятся не только современные 32-битные микроконтроллеры (ARM Limited, , но и 16 (TI), и даже 8-битные (Intel,Atmel). Внутри каждого семейства часто можно встретить почти одинаковые модели, различающиеся скоростью работы ЦПУ и объемом памяти.

Наиболее популярными семействами микроконтроллеров являются следующие :

ARM –32- и 64-битные микроконтроллеры с архитектурой, в которой быстродействие увеличивается за счёт упрощения инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим. Основные производители: AMD, Apple, Analog Devices, Atmel, Xilinx, Altera, Cirrus Logic, Intel (до 27 июня 2006 года), Marvell, NXP, STMicroelectronics, Samsung, LG, MediaTek, MStar, Qualcomm, Sony, Texas Instruments, nVidia, Freescale, Миландр, HiSilicon. В последнее время наибольшую распространённость имеет ARM CortexM.

AVR – семейство 8-битных микроконтроллеров. Представителем AVR микроконтроллеров является известная фирма Atmel. Исходя из необходимости выполняемых задач, различают следующие семейства: AVR UC3; AVR XMEGA; megaAVR; tinyAVR.

MCS 51 (Intel 8051) – однокристальные, 8-разрядные микроконтроллеры гарвардской архитектуры, которые были впервые произведены Intel в 1980 году для использования во встраиваемых системах. В течение 1980-х и начале 1990-х годов был чрезвычайно популярен, однако позже устарел и был вытеснен более современными устройствами, также с 8051-совместимыми ядрами, производимыми более чем 20 независимыми производителями, такими, как Atmel, Maxim IC (дочерняя компания Dallas Semiconductor), NXP , Winbond, Silicon Laboratories, Texas Instruments и Cypress Semiconductor).

PIC — серия микроконтроллеров, имеющих разделение хранилищ и каналы инструкций и данных (гарвардская архитектура). Microchip Technology Inc является изготовителем МК PIC, которые бывают 8-, 16- и 32-битные. Используются в основном для расширения периферийных возможностей ввода-вывода микропроцессоров.

В последнее время широкое распространение получила торговая марка аппаратно-программных средств для построения простых систем автоматики и робототехники, ориентированная на непрофессиональных пользователей, под названием **Arduino**, что представляет собой бесплатную программную оболочку в программной части и набор смонтированных печатных плат в аппаратной части с полностью открытой архитектурой системы, т.е. свободное копирование и пополнение линейки продукции Arduino. В линейке устройств Arduino в основном применяются микроконтроллеры Atmel AVR ATmega328, ATmega168, ATmega2560, ATmega32U4, ATTiny85 с частотой тактирования 16 или 8 МГц.

На рис. 2.12.2 приведена структурная схема 8-разрядного микроконтроллера архитектуры Intel 8051. На рис. 2.12.3 приведена структурная схема МК ADuC812 в составе учебного стенда SDK-1.1.

Микроконтроллер характеризуется большим числом параметров, поскольку он одновременно является сложным программно-управляемым устройством и электронным прибором (микросхемой). Приставка «микро» в названии микроконтроллера означает, что выполняется он по микроэлектронной технологии.

В ходе работы микроконтроллер считывает команды из памяти или порта ввода и исполняет их. Назначение каждой команды, определяется **системой команд** микроконтроллера, которая заложена в архитектуре микроконтроллера, и выполнение кода команды выражается в проведении внутренними элементами микросхемы определенных микроопераций.

Микроконтроллеры позволяют гибко управлять различными электронными и электрическими устройствами и, как правило, не работают в одиночку, а запаиваются в схему, где, кроме него, подключаются экраны, клавиатурные входы и различные датчики.

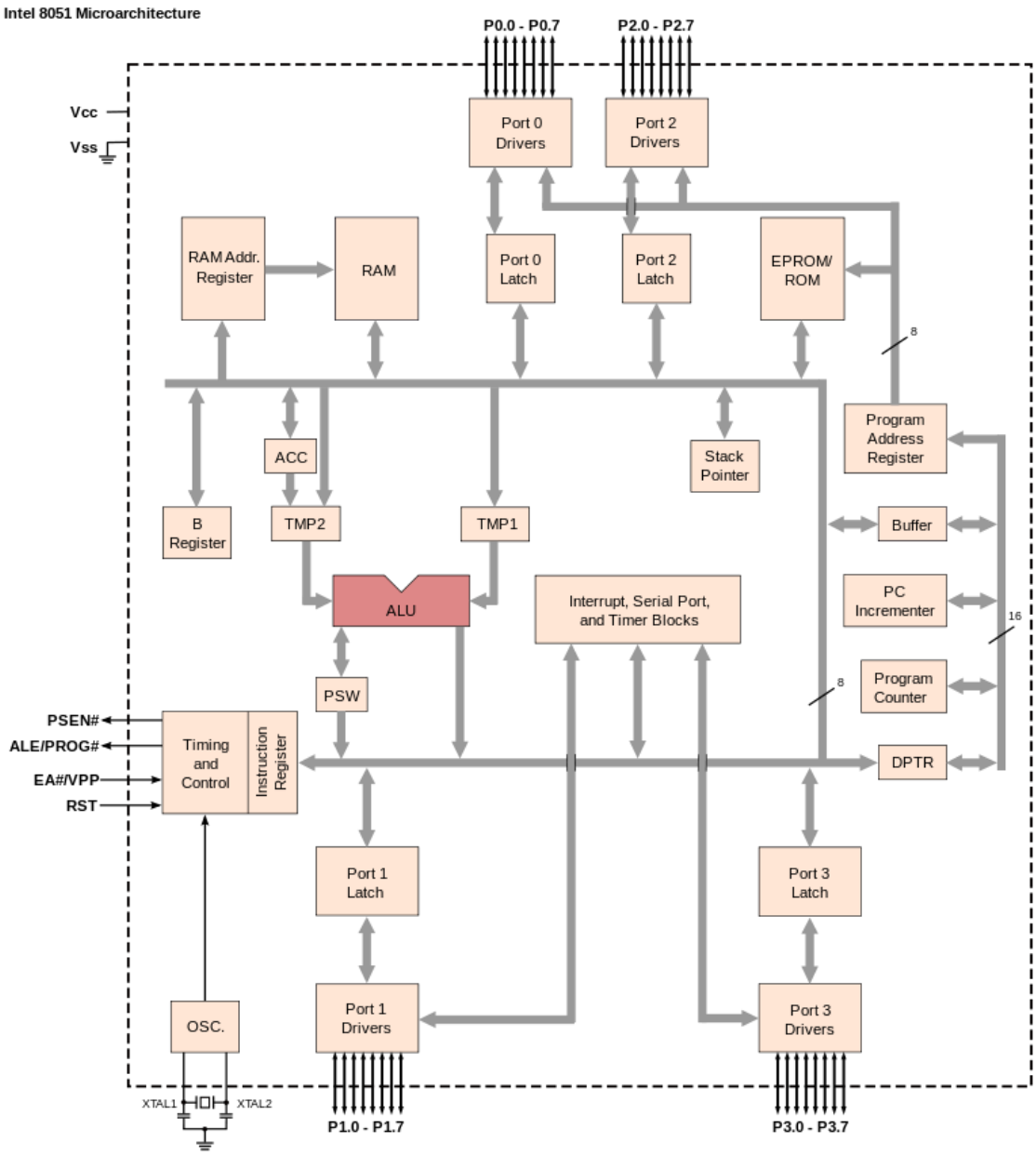


Рис. 2.12.2 Структурная схема архитектуры Intel 8051

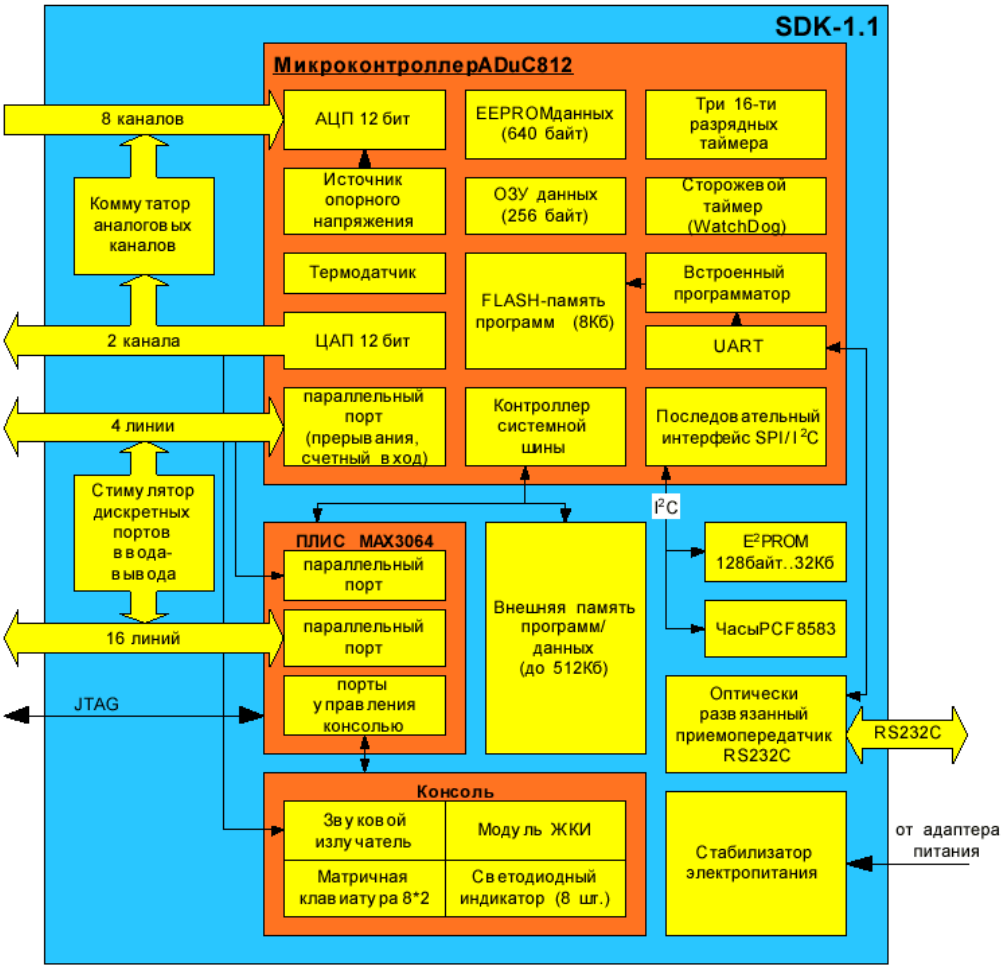


Рис. 2.12.3 Структурная схема МК ADuC812 в составе SDK-1.1

3 ПРИНЦИПЫ ОРГАНИЗАЦИИ ПОДСИСТЕМЫ ПАМЯТИ ЭВМ

3.1 Иерархическая структура памяти ЭВМ

Памятью ЭВМ называется совокупность устройств, предназначенных для запоминания, хранения и выдачи информации.

Основными характеристиками отдельных устройств памяти (запоминающих устройств) являются емкость памяти, быстродействие и стоимость хранения единицы информации (бита).

Емкость памяти определяется максимальным количеством данных, которые могут в ней храниться. Ёмкость измеряется в двоичных единицах (битах), машинных словах, но большей частью в байтах (1 байт = 8 бит). Часто емкость памяти выражают через число К = 2^{10} = 1024, например, 1024 бит = Кбит (килобит), 1024 байт = Кбайт (килобайт), 1024 Кбайт = 1 Мбайт (мегабайт), 1024 Мбайт = 1 Гбайт (гигабайт), 1024 Гбайт = 1 Тбайт (терабайт).

Быстродействие (задержка) памяти определяется **временем доступа** и **длительностью цикла памяти**. Время доступа представляет собой промежуток времени между выдачей запроса на чтение и моментом поступления запрошенного слова из памяти. Длительность цикла памяти определяется минимальным временем между двумя последовательными обращениями к памяти.

Требования к увеличению емкости и быстродействия памяти, а также к снижению ее стоимости являются **противоречивыми.** Чем больше быстродействие, тем технически труднее достигается и дороже обходится увеличение емкости памяти. Исходя из этого, память ЭВМ организуется в виде **иерархической структуры** запоминающих устройств, обладающих различным быстродействием, емкостью и стоимостью. Причем более высокий уровень меньше по объему, быстрее и имеет большую стоимость в пересчёте на байт, чем более низкий уровень. Уровни иерархии взаимосвязаны: все данные на одном уровне могут быть также найдены на низком уровне, и все данные на этом (более низком) уровне могут быть найдены на следующем, нижележащем уровне, и так далее, пока мы не достигнем основания иерархии. В структуре памяти, представленной на рис. 3.1, к верхнему (сверхоперативному) уровню относятся: управляющая память, регистры различного назначения, стек регистров, буферная память. На втором уровне находится основная (или оперативная) память. На последующих уров-

нях размещается внешняя и архивная память. Система управления памятью обеспечивает обмен информационными блоками между уровнями, причем обычно первое обращение к блоку информации вызывает его перемещение с низкого (медленного) уровня на более высокий. Это позволяет при последующих обращениях к данному блоку осуществлять его выборку с более быстродействующего уровня памяти.

Успешное или неуспешное обращение к более высокому уровню называется соответственно «попаданием» (hit) или «промахом» (miss). Попадание есть обращение к объекту в памяти, который найден на более высоком уровне, в то время как промах означает, что он не найден на этом уровне. Доля попаданий, или коэффициент попаданий, есть доля обращений, найденных на более высоком уровне. Иногда она представляется в процентах. Аналогично для промахов.

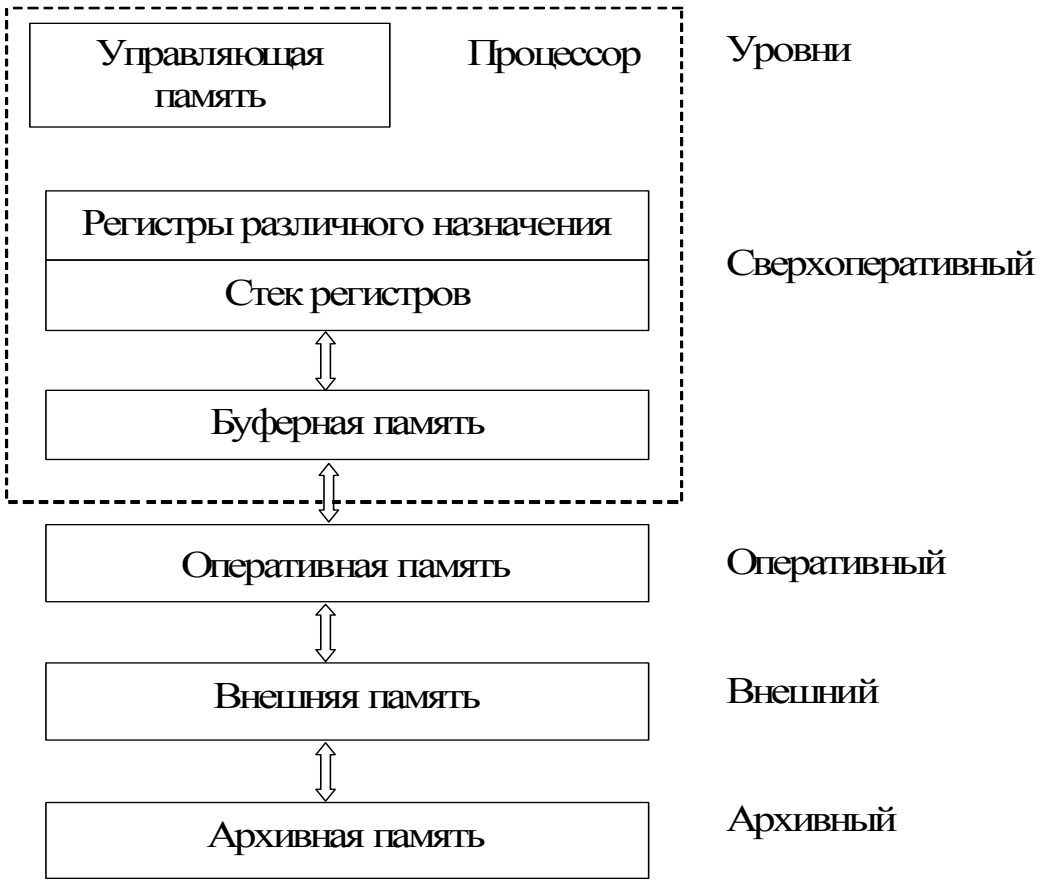


Рис. 3.1. Иерархическая структура памяти

Сравнительно небольшая емкость оперативной памяти компенсируется практически неограниченной емкостью внешних запоминающих устройств. Однако эти устройства работают намного медленнее, чем

оперативная память. Время обращения за данными для магнитных дисков составляет десятки микросекунд. Для сравнения: цикл обращения к оперативной памяти (ОП) составляет несколько десятков наносекунд. Исходя из этого, вычислительный процесс должен протекать с возможно меньшим числом обращений к внешней памяти. Память современных компьютеров реализуется на микросхемах статических и динамических запоминающих устройств с произвольной выборкой. Микросхемы статических ЗУ (SRAM) имеют меньшее время доступа и не требуют циклов регенерации (восстановления) информации. Микросхемы динамических ЗУ (DRAM) характеризуются большей емкостью и меньшей стоимостью, но требуют схем регенерации и имеют значительно большее время доступа. У статических ЗУ время доступа совпадает с длительностью цикла.

По этим причинам в основной памяти практически любого компьютера, проданного после 1975 г., использовались полупроводниковые микросхемы DRAM (SDRAM, DDR SDRAM, RDRAM). Для построения кэш-памяти применяются SRAM.

Непрерывный рост производительности ЭВМ проявляется в первую очередь в повышении скорости работы процессора. Быстродействие ОП также растет, но все время отстает от быстродействия аппаратных средств процессора в значительной степени потому, что одновременно происходит опережающий рост её емкости, что делает более трудным уменьшение времени цикла работы памяти. Вследствие этого быстродействие ОП часто оказывается недостаточным для обеспечения требуемой производительности ЭВМ. Это проявляется в несоответствии пропускных способностей процессора и ОП. Возникающая проблема выравнивания их пропускных способностей решается путем использования сверхоперативной буферной памяти небольшой емкости и повышенного быстродействия, хранящей команды и данные, относящиеся к обрабатываемому участку программы.

При обращении к блоку данных, находящемуся на оперативном уровне, его копия пересылается в сверхоперативную буферную память (СБП). Последующие обращения производятся к копии блока данных, находящейся в СБП. Поскольку время выборки из сверхоперативной буферной памяти $t_{СБУ}$ (несколько наносекунд) много меньше времени выборки из оперативной памяти $t_{ОП}$, введение в структуру памяти СБП приводит к уменьшению эквивалентного времени обращения $t_Э$ по сравнению с $t_{ОП}$:

$$t_Э = t_{СБП} + \alpha t_{ОП} ,$$

где $\alpha = (1 - q)$ и q – вероятность нахождения блока в СБП в момент обращения к нему, т.е. вероятность «попадания». Очевидно, что при высокой вероятности попадания эквивалентное время обращения приближается к времени обращения к СБП.

В основе такой организации взаимодействия ОП и СБП лежит принцип локальности обращений, согласно которому при выполнении какой-либо программы (практически для всех классов задач) большая часть обращений в пределах некоторого интервала времени приходится на ограниченную область адресного пространства ОП, причем обращения к командам и элементам данных этой области производятся многократно. Это позволяет копии наиболее часто используемых участков программ и некоторых данных загрузить в СБП и таким образом обеспечить высокую вероятность попадания q. Высокая эффективность применения СБП достигается при $q \geq 0{,}9$.

Буферная память не является программно доступной. Это значит, что она влияет только на производительность ЭВМ, но не должна оказывать влияния на программирование прикладных задач. Поэтому она получила название **кэш-памяти** (в переводе с английского – тайник). В современных компьютерах применяют многоуровневую кэш-память (до трех уровней), которая еще больше способствует производительности ЭВМ. Как правило, на первом уровне используются раздельные кэш-памяти для команд и данных, а на других уровнях данные и команды хранятся в одних и тех же кэш-памятях.

При работе с памятью в основу кладется принцип **локальности.** Этот принцип гласит, что программы в любой момент времени обращаются к относительно небольшой части своего адресного пространства. Существуют две разновидности локальности:

Локальность, связанная со временем: принцип, в котором утверждается, что если было обращение к элементу данных, то высока вероятность, что к нему скоро последует новое обращение.

Локальность, связанная с пространством: принцип, в котором утверждается, что если было обращение к элементу данных, то высока вероятность, что скоро последует обращение и к элементам данных по соседним адресам.

3.2 Организация стека регистров

Регистровая структура процессора была рассмотрена в разд. 2.9.

Стек регистров, реализующий безадресное задание операндов, является эффективным элементом архитектуры ЭВМ. Стек представляет

собой группу последовательно пронумерованных регистров, снабжённых указателем стека, в котором автоматически при записи устанавливается номер первого свободного регистра стека (вершина стека). Существует два основных способа организации стека регистров:

LIFO (Last-in First-Out) – последний пришёл – первый ушёл;

FIFO (First-in First-Out) – первый пришёл – первый ушёл.

Механизм стековой адресации для первого способа поясняется на рис. 3.2. Для реализации адресации по способу LIFO используется счётчик адреса СЧА, который перед началом работы устанавливается в состояние ноль, и память (стек) считается пустой. Состояние СЧА определяет адрес первой свободной ячейки. Слово загружается в стек с входной шины *X* в момент поступления сигнала записи ЗП.

По сигналу ЗП слово *X* записывается в регистр Р[СЧА], номер которого определяется текущим состоянием счётчика адреса, после чего с задержкой *D*, достаточной для выполнения микрооперации записи Р[СЧА]:=*X*, состояние счетчика увеличивается на единицу. Таким образом, при последовательной загрузке слова *A*, *B* и *C* размещаются в регистрах с адресами Р[*S*], Р[*S* + 1] и Р[*S* + 2], где *S* – состояние счётчика на момент начала загрузки. Операция чтения слова из ЗУ инициируется сигналом ЧТ, при поступлении которого состояние счётчика уменьшается на единицу, после чего на выходную шину *Y* поступает слово, записанное в стек последним. Если слова загружались в стек в порядке *A*, *B*, *C*, то они могут быть прочитаны только в обратном порядке: *C*, *B*, *A*.

В современных архитектурах процессоров стек и стековая адресация широко используются при организации переходов к подпрограммам и возврата из них, а также в системах прерывания.

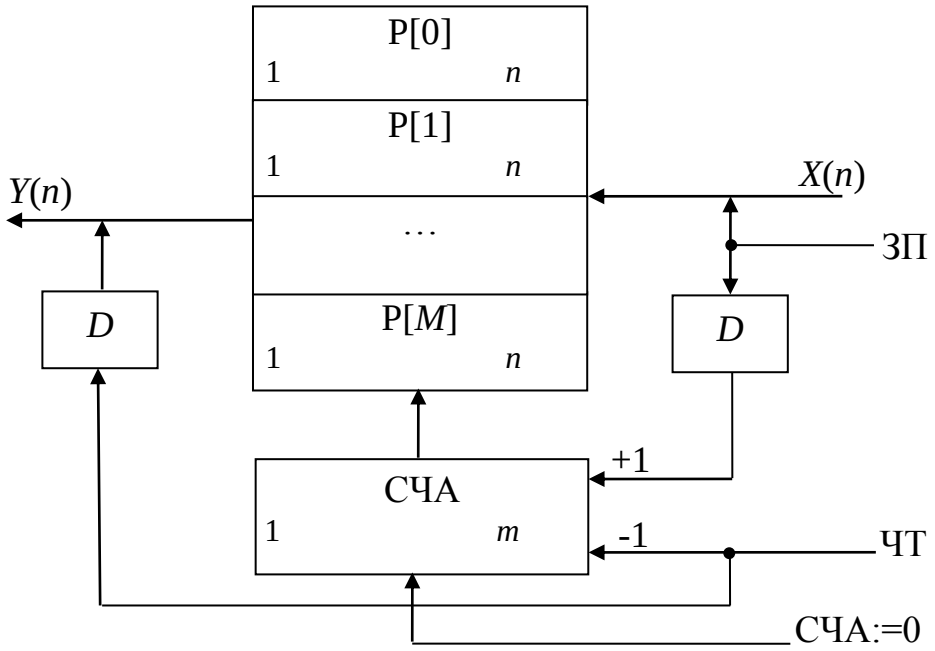


Рис. 3.2. Механизм стековой адресации по способу LIFO

3.3 Способы организации кэш-памяти

Общие сведения

В функциональном отношении кэш-память рассматривается как буферное ЗУ, размещённое между основной (оперативной) памятью и процессором. Основное назначение кэш-памяти – кратковременное хранение и выдача активной информации процессору, что сокращает число обращений к основной памяти, скорость работы которой меньше, чем кэш-памяти.

За единицу информации при обмене между основной памятью и кэш-памятью принята строка (линейка), причём под строкой понимается набор слов, выбираемый из оперативной памяти при одном к ней обращении. Хранимая в оперативной памяти информация представляется, таким образом, совокупностью строк с последовательными адресами. В любой момент времени строки в кэш-памяти представляют собой копии строк из некоторого их набора в ОП, однако расположены они необязательно в такой же последовательности, как в ОП.

Построение кэш-памяти может осуществляться по различным принципам, которые будут рассмотрены ниже. Но общим для всех способов построения кэш-памяти является использование так называемых адресных тегов. **Адресный тег –** это расширенный адрес, который объ-

единяет адреса всех слов, принадлежащих данной строке. Он указывает, какую строку в ОП представляет данная строка в кэш-памяти.

3.3.1 Типовая структура кэш-памяти

Рассмотрим типовую структуру кэш-памяти (рис. 3.3), включающую основные блоки, которые обеспечивают её взаимодействие с ОП и центральным процессором.

Строки, составленные из информационных слов, и связанные с ними адресные теги хранятся в накопителе, который является основой кэш-памяти, остальные блоки относятся к кэш-контроллеру. Адрес требуемого слова, поступающий от центрального процессора (ЦП), вводится в блок обработки адресов, в котором реализуются принятые в данной кэш-памяти принципы использования адресов при организации их сравнения с адресными тегами. Само сравнение производится в блоке сравнения адресов (БСА), который конструктивно совмещается с накопителем, если кэш-память строится по схеме ассоциативной памяти. Назначение БСА состоит в выявлении попадания или промаха при обработке запросов от центрального процессора. Если имеет место кэш-попадание (совпадение теговой части адреса, поступающего от центрального процессора, с адресным тегом одной из ячеек кэш-памяти), то в режиме чтения информации соответствующая строка из кэш-памяти переписывается в регистр строк. С помощью селектора из неё выделяется искомое слово, которое и направляется в центральный процессор.

В случае промаха с помощью блока формирования запросов осуществляется инициализация выборки из ОП необходимой строки.

Адресация ОП при этом производится в соответствии с информацией, поступившей от центрального процессора. Выбираемая из памяти строка вместе со своим адресным тегом помещается в накопитель и регистр строк, а затем искомое слово передается в центральный процессор.

В режиме записи информации в память адрес обрабатывается также, как и при чтении. Само же слово информации из ЦП проходит через демультиплексор и заносится в регистр строк. Далее, в зависимости от выбранного способа записи оно может загрузиться в накопитель строк кэш-памяти и в ОП или только в кэш-память.

Для высвобождения места в кэш-памяти с целью записи выбираемой из ОП строки одна из строк удаляется. Определение удаляемой строки производится посредством блока замены строк, в котором хранится информация, необходимая для реализации принятой стратегии обновления находящихся в накопителе строк.

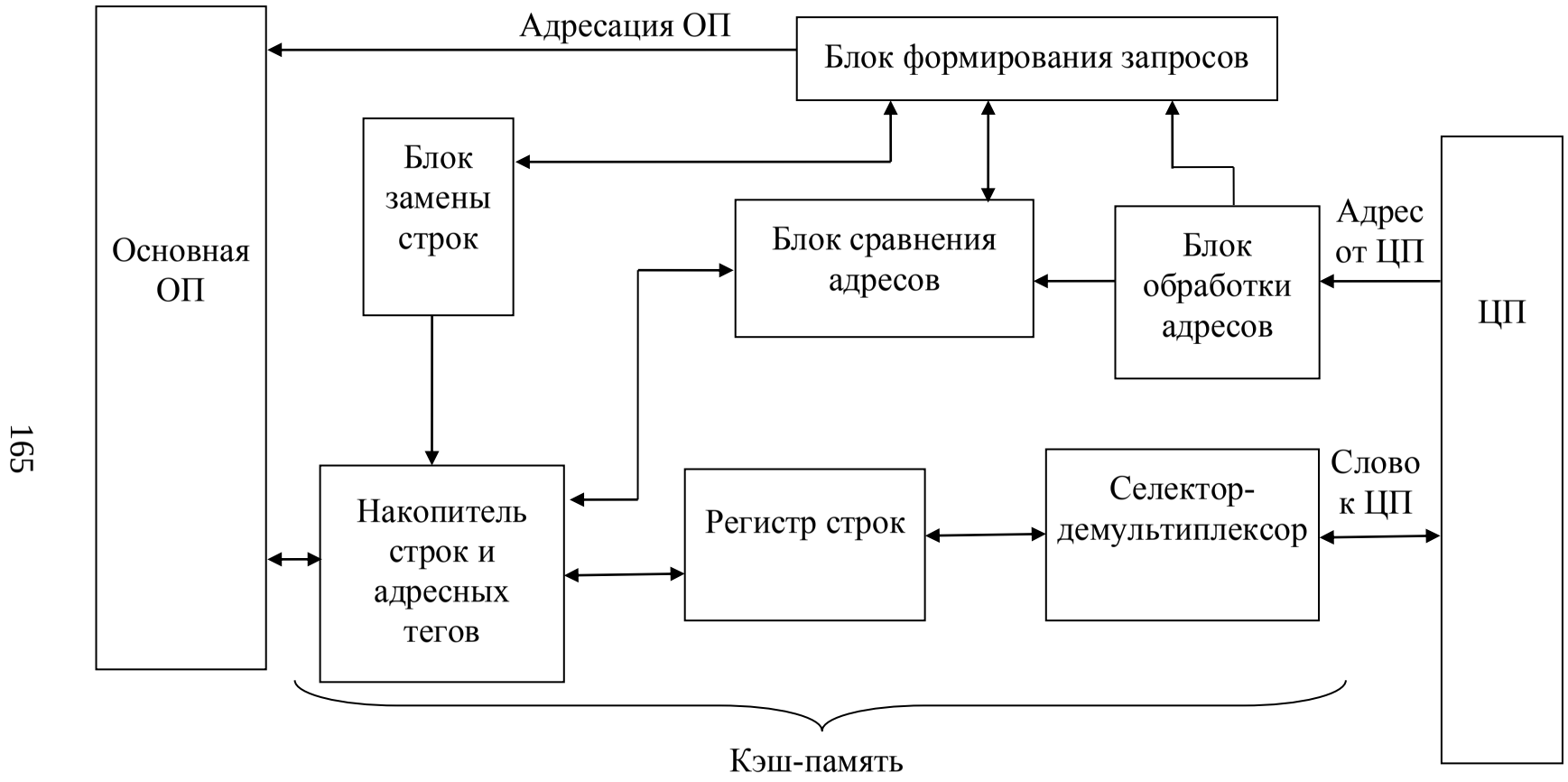


Рис. 3.3. Типовая структура кэш-памяти

3.3.2 Способы размещения данных в кэш-памяти

Существует три основных способа размещения данных в кэш-памяти: **прямое распределение, полностью ассоциативное** распределение и **частично ассоциативное** распределение. Ниже подробно описан каждый способ размещения и его механизм преобразования адресов. Для того чтобы конкретизировать описание, положим, что кэш-память может содержать 1024 строки, размер строки – 16 слов, а основная память может содержать 16384 строк. Для адресации основной памяти используется 32 бита, из них старшие 30 показывают адрес строки, а младшие 2 бит – адрес слова внутри этой строки (байтовое смещение). При одном обращении к памяти выбирается одна строка; 1024 строки кэш-памяти указываются 10-разрядными адресами.

Если рассмотреть принцип работы кэша на примере библиотеки с полками и книжным столом, то стол является кэшом — укромным местом для хранения вещей (книг), которые нужно изучить. На рис. 3.4 показана простая кэш-память до и после запроса элемента данных, который изначально в ней отсутствовал. Перед запросом в кэш-памяти находилась коллекция данных, на которые были последние ссылки X_1, X_2,..., X_n. а процессор запросил слово X_n, которого в кэш памяти нет. Этот запрос оказался неудачным, и слово X_n было перенесено из оперативной памяти в кэш.

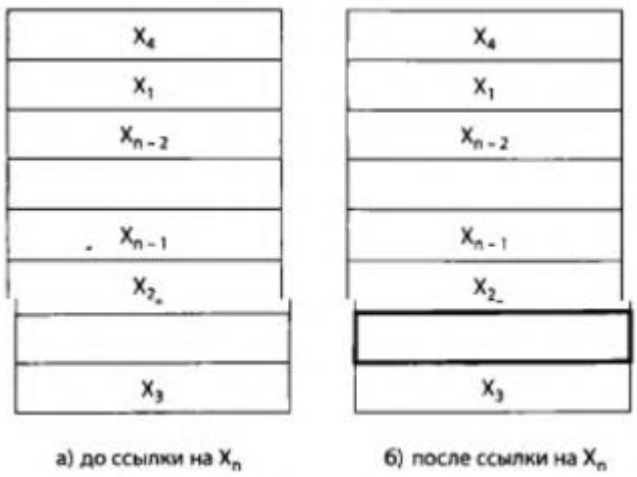


Рис. 3.4. Кэш-память до и после обращения к слову X_n

Если каждое слово может попадать во вполне определенное одно и то же место в кэш-памяти, то найти слово, находящееся в кэш-памяти, будет нетрудно. Наиболее простым способом определения места в кэш-памяти для каждого слова является определение места в кэш-памяти на основе адреса слова в памяти. Такая структура кэша называется **непо-**

средственно отображенной, поскольку каждое место в памяти отображается непосредственно на одно место в кэш-памяти.

Прямое распределение (непосредственное отображение)

Обычное отображение между адресами и местами в кэше для кэш-памяти с непосредственным отображением, как правило, устроено довольно просто. Например, почти во всех устройствах кэш-памяти с непосредственным отображением для поиска блока используется следующее проецирование:

(Адрес блока) modulo (Количество блоков в кэш-памяти)

Если количество записей в кэш-памяти является степенью числа 2, то модуль (modulo) может быть вычислен просто за счет использования младших $\log_2$ (размер кэш-памяти в блоках) разрядов адреса. Таким образом, кэш память из восьми блоков использует три самых младших разряда ($8 = 2^3$) адреса блока. Например, на рис. 3.5 показано, как память с адресами между 1_{10} (00001_2) и 29_{10} (11101_2) отображается на места 1_{10} (001_2) и 5_{10} (101_2) в кэш-памяти с непосредственным отображением, состоящей из восьми слов.

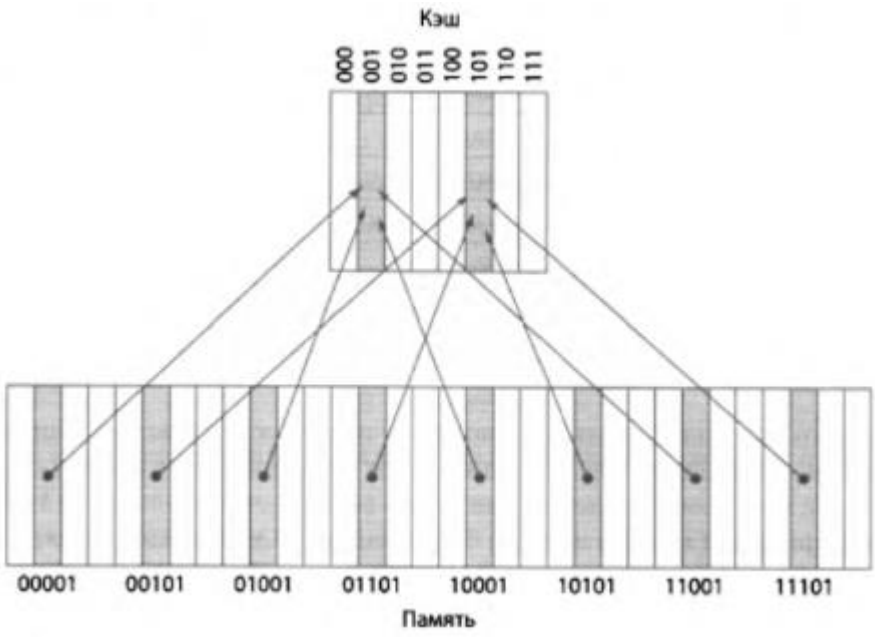


Рис. 3.5. Кэш-память с прямым распределением, имеющая 8 записей с указанием адресов слов памяти в диапазоне от 0 до 31, отображаемых одни и те же места в кэш-памяти. .

Поскольку в кэш-памяти 8 слов, адрес X проецируется на имеющееся в кэш-памяти с непосредственным отображением слово X по модулю 8. То есть младшие $\log_2(8) = 3$ разряда используются в качестве индекса кэш-памяти.

Таким образом, адреса 00001_2. 01001_2,. 10001_2 и 11001_2 проецируются на запись 001_2 кэш- памяти, а адреса 00101_2, 01101_2, 10101_2 и 11101_2 проецируются на запись 101_2 кэш-памяти.

Обращение к кэш-памяти

В таблице 3.1 показана последовательность из девяти обращений к памяти, приходящихся на пустую кэш-память, состоящую из восьми блоков, включая действие, предпринимаемое при каждом обращении (табл. 3.1). На рис. 3.6 показано, как содержимое кэш памяти изменяется после каждого промаха. Поскольку в кэш памяти восемь блоков, самые младшие три разряда адреса дают номер блока:

Десятичный адрес обращения	**Двоичный адрес обращения**	**Попадание или промах**	**Заданный блок кэш-памяти (где найдены или куда помещены данные)**
22	10110_2	промах (5.6, б)	$(10110_2 \bmod 8) = 110_2$
26	11010_2	промах (5.6, а)	$(11010_2 \bmod 8) = 010_2$
22	10110_2	попадание	$(10110_2 \bmod 8) = 110_2$
26	11010_2	попадание	$(11010_2 \bmod 8) = 010_2$
16	10000_2	промах (5.6, г)	$(10000_2 \bmod 8) = 000_2$
3	00011_2	промах (5.6, д)	$(10011_2 \bmod 8) = 011_2$
16	10000_2	попадание	$(10000_2 \bmod 8) = 000_2$
18	10010_2	промах (5.6, е)	$(10010_2 \bmod 8) = 010_2$
16	10000_2	попадание	$(10000_2 \bmod 8) = 000_2$

Из-за того, что кэш память не заполнена, несколько первых обращений будут неудачными. На восьмом обращении для блока возникает конфликтное требование. Слово по адресу 18 (10010_2) должно быть помешено в блок кэш-памяти 2 (010_2). Следовательно, оно должно заменить слово из адреса 26 (11010_2), которое уже находится в блоке кэш-памяти 2 (010_2). Такое поведение позволяет кэш памяти воспользоваться локальностью, связанной со временем: слова, обращение к которым состоялось недавно, заменяют те слова, обращение к которым было чуть раньше.

Если вернуться к примеру с библиотекой, то эта ситуация является прямой аналогией с необходимостью наличия книги с полки и необходимостью дополнительного места на столе — какая-то книга, которая уже на нем лежит, должна быть возвращена на полку. В кэш-памяти с непосредственным отображением есть только одно место, куда можно поместить только что запрошенный элемент, и, следовательно, только один вариант замены.

Индекс	V	Тег	Данные
000	Нет		
001	Нет		
010	Нет		
011	Нет		
100	Нет		
101	Нет		
110	Нет		
111	Нет		

а) начальное состояние кэш-памяти после подачи питания

Индекс	V	Тег	Данные
000	Нет		
001	Нет		
010	Нет		
011	Нет		
100	Нет		
101	Нет		
110	Да	10_2	Память (10110_2)
111	Нет		

б) после обработки неудачи по адресу (10110_2)

Индекс	V	Тег	Данные
000	Нет		
001	Нет		
010	Да	11_2	Память (11010_2)
011	Нет		
100	Нет		
101	Нет		
110	Да	10_2	Память (10110_2)
111	Нет		

в) после обработки неудачи по адресу (11010_2)

Рис. 3.6. а,б,в Содержимое кэш-памяти после каждого неудачного обращения.

Индекс	V	Тег	Данные
000	Нет	10_2	Память (10000_2)
001	Нет		
010	Да	11_2	Память (11010_2)
011	Нет		
100	Нет		
101	Нет		
110	Да	10_2	Память (10110_2)
111	Нет		

г) после обработки неудачи по адресу (10000_2)

Индекс	V	Тег	Данные
000	Нет	10_2	Память (10000_2)
001	Нет		
010	Да	11_2	Память (11010_2)
011	Нет	00_2	Память (00011_2)
100	Нет		
101	Нет		
110	Да	10_2	Память (10110_2)
111	Нет		

д) после обработки неудачи по адресу (00011_2)

Индекс	V	Тег	Данные
000	Нет	10_2	Память (10000_2)
001	Нет		
010	Да	10_2	Память (10010_2)
011	Нет	00_2	Память (00011_2)
100	Нет		
101	Нет		
110	Да	10_2	Память (10110_2)
111	Нет		

е) после обработки неудачи по адресу (10010_2)

Рис. 3.6. г,д,е. Содержимое кэш-памяти после каждого неудачного обращения.

Нам известно, куда обращаться в кэш-намяти для каждого возможного адреса: самые младшие разряды адреса могут быть использованы для поиска уникальной записи в кэш-памяти, на которую мог быть спроецирован адрес. На рис. 5.7 показано, как адрес, к которому идет обращение, делится на:

- поле тега, которое используется для сравнения со значением поля тега кэш¬памяти;
- индекс кэш-памяти, который используется для выбора блока.

Индекс блока кэш-памяти наряду с содержимым тега этого блока уникальным образом определяют адрес в памяти того слова, которое содержится в блоке кэш-памяти. Поскольку поле индекса используется в качестве адреса для обращения к кэш-памяти и поскольку поле, состоящее из *n* разрядов, имеет 2^n значений, общее количество записей в кэш-памяти с непосредственным отображением должно быть кратным степени числа 2. В MIPS-архитектуре, поскольку слова выровнены таким образом, чтобы быть кратными четырем байтам, самые младшие два разряда каждого адреса определяют байт внутри слова. Следовательно, самые младшие два разряда при выборе слова в блоке игнорируются.

Общее количество разрядов, необходимых для кэш-памяти, определяется функцией от размера кэш-памяти и размера адресного пространства, поскольку кэш-память включает в себя как место для хранения данных, так и теги. Размером рассмотренного ранее блока было одно слово, но обычно блок состоит из нескольких слов.

Для следующей ситуации:

- 32-разрядный байтовый адрес;
- кэш-память с непосредственным отображением;
- размер кэш памяти в 2я блоков, следовательно, для индекса используются n бит;
- размер блока 2^m слов (2^{m+2} байт), следовательно, для слова внутри блока используются *m* разрядов, и два разряда используются для байтовой части адреса

При этом, размер ноля тега составляет $32 – (n + m + 2)$ бит.

Общее количество разрядов в кэш памяти с прямым распределением составляет:

$$2^n \text{ x (размер блока + размер тега + размер поля достоверности).}$$

Поскольку размер блока равен 2^m слов и нам нужен один бит для поля достоверности (V), количество бит в такой кэш памяти будет равно:

$$2^n \text{ x } (2^m \text{ x } 32 + (32–n–m–2) + 1) –2^n \text{ x } (2^m \text{ x } 32 + 31 –n–m).$$

Хотя здесь представлен реальный размер в битах, соглашение о названиях требует исключить размер тега и поля достоверности и учитывать только размер данных. Поэтому кэш-память на рис.3.7 называется кэш-памятью размером 4 Кбайт.

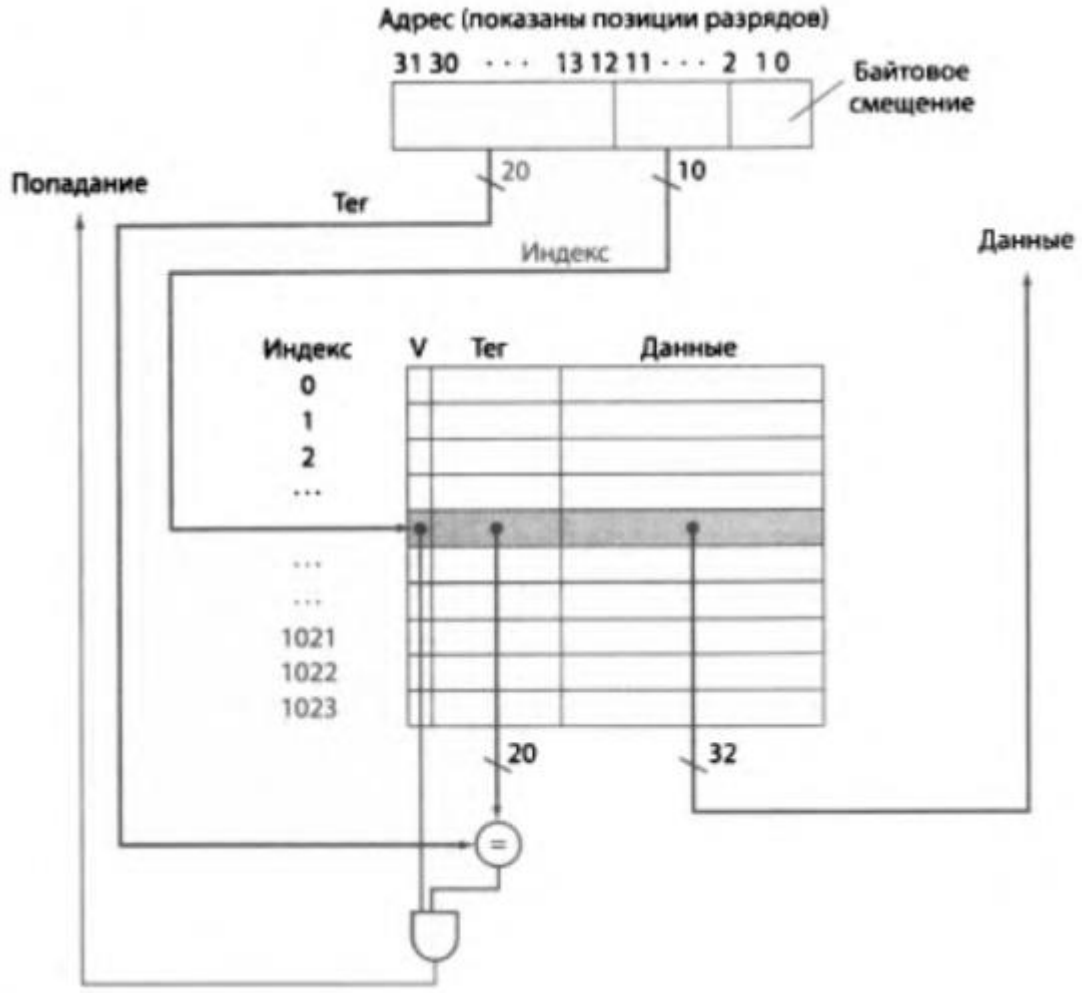


Рис. 3.7. Прямая адресация кэш-памяти.

Кэш-память на рис. 3.7 сохраняет 1024 слов или 4 Кбайт. Тег, имеющийся в кэш-памяти, сравнивается со старшей частью адреса для определения, соответствует ли запись в кэш-памяти запрошенному адресу. Поскольку кэш-память содержит 2^{10} (или 1024) слов и размер блока составляет одно слово, для индекса кэш-памяти используются 10 разрядов, оставляя 32 – 10 – 2 = 20 разрядов для сравнения с тегом. Если тег и старшие 20 разрядов адреса равны друг другу и бит достоверности установлен, значит, обращение к кэш-памяти было успешным, и слово предоставляется процессору. В противном случае обращение заканчивается неудачей.

Полностью и частично ассоциативное распределение

При **полностью ассоциативном** распределении (fully associative) допускается размещение каждой строки основной памяти на месте любой строки кэш-памяти. Для поиска нужного блока в полностью ассоци-

ативной кэш-памяти должны быть просмотрены все имеющиеся в ней записи, потому что блок может быть помещен в любую из них. Для удобства поиск ведется параллельно с использованием компаратора, связанного с каждой записью кэш-памяти. Компараторы существенно повышают стоимость компьютерного оборудования, из-за чего полностью ассоциативное размещение остается практичным только для кэш-памяти с небольшим количеством блоков.

Между конструкциями с непосредственным отображением и полной ассоциативностью существует промежуточный вариант, называемый **частично-ассоциативным** (множественно-ассоциативным). В кэш-памяти с частичной ассоциативностью существует вполне определенное число мест, в которые может быть помещен каждый блок. Кэш память с n местами для каждого блока называется n-канальной ассоциативной кэш-памятью. Эта кэш-память состоит из группы наборов, каждый из которых состоит из n блоков. Каждый блок в памяти отображается на уникальный набор в кэш-памяти, задаваемый полем индекса, и блок может быть помещен в любой элемент этого набора. Таким образом, в частично-ассоциативном размещении комбинируется размещение с непосредственным отображением и полностью ассоциативное размещение: блок непосредственно отображается на конкретный набор, а затем все блоки в наборе подвергаются поиску на соответствие. Например, на рис. 3.8 показано, куда может быть помещен блок 12 в кэш памяти, состоящей всего из восьми блоков, согласно трем правилам размещения блоков.

Следует напомнить, что в кэш-памяти с непосредственным отображением позиция блока памяти задается следующим образом:

(Номер блока) mod (Количество блоков в кэш-памяти)

В кэш-памяти с частичной ассоциативностью набор, содержащий блок памяти, задается следующим образом:

(Номер блока) mod (Количество наборов в кэш-памяти)

Поскольку блок может быть помещен в любой элемент набора, должны быть просмотрены все теги всех элементов набора. В полностью ассоциативной кэш-памяти блок может быть помещен в любое место, и должны быть просмотрены все теги всех блоков кэш памяти.

Все стратегии размещения всех блоков могут быть также представлены как вариации частичной ассоциативности. На рис. 3.9 показаны возможные ассоциативные структуры для кэш памяти, состоящей из восьми блоков.

Рис. 3.8. Размещение блока памяти с адресом 12 в кэш-памяти из 8 блоков.

Размещение блока памяти с адресом 12 в кэш-памяти из восьми блоков различается для кэш-памяти с непосредственным отображением, частичной ассоциативностью и полной ассоциативностью. В размещении с непосредственным отображением, существует только один блок кэш-памяти, где может быть найден блок памяти 12, и этот блок - определяется как (12 по модулю 8) = 4 В двухканальной частично-ассоциативной кэш-памяти будет четыре набора, и блок памяти 12 должен быть в наборе (12 по модулю 4) = 0; блок памяти и может быть в любом элементе набора В размещении с полной ассоциативностью блок памяти с адресом 12 может появиться в любом из восьми блоков кэш-памяти.

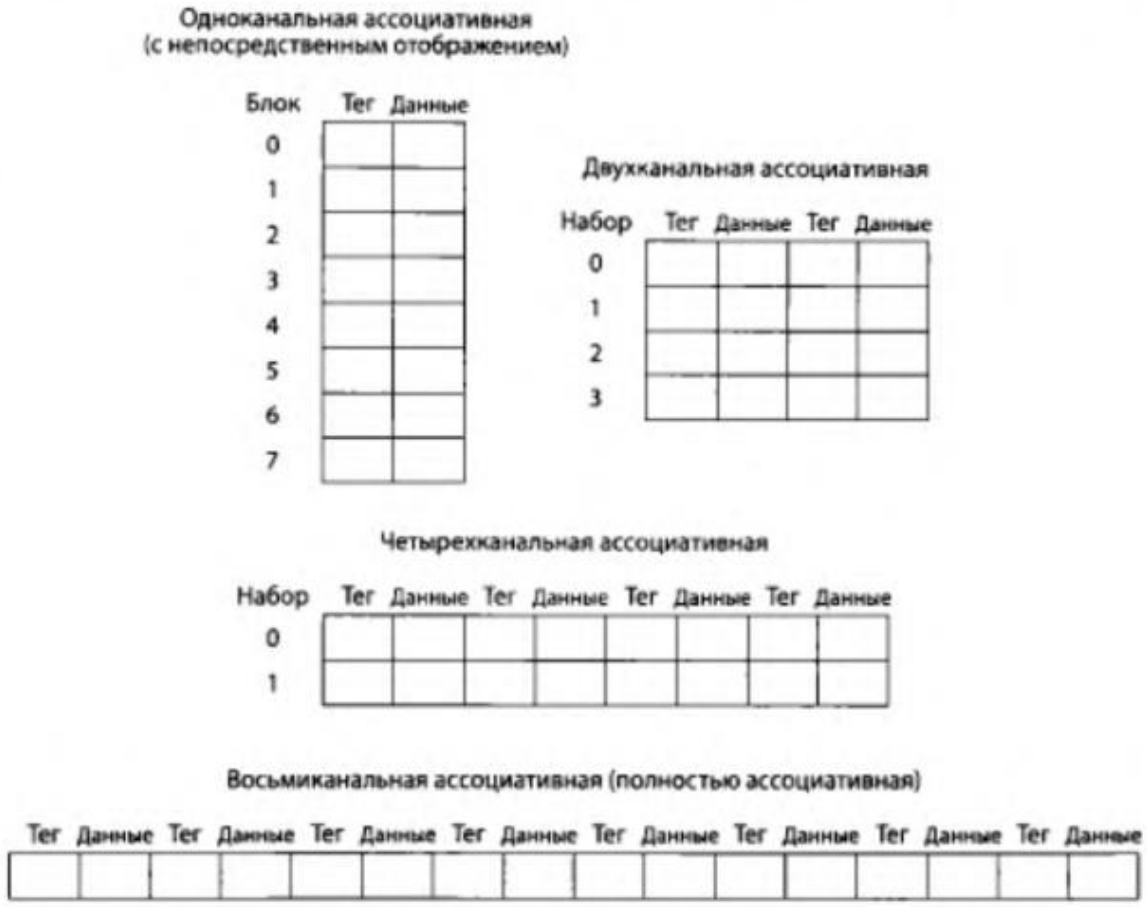


Рис. 3.9. Кэш-память из восьми блоков, сконфигурированная с непосредственным отображением, с двухканальной ассоциативностью, с четырехканальной ассоциативностью и в полностью ассоциативном варианте

Общий размер кэш-памяти в блоках равен количеству наборов, помноженному на количество каналов ассоциативности. Таким образом, для фиксированного размера кэш-памяти увеличение количества каналов ассоциативности уменьшает количество наборов при увеличении количества элементов в наборе. При наличии восьми блоков восьмиканальная ассоциативная кэш-память становится аналогом полностью ассоциативной кэш-памяти.

3.3.3 Методы обновления строк основной памяти и кэша

В табл. 5.1 приведены условия сохранения и обновления информации в ячейках кэш-памяти и основной памяти.

Если процессор намерен получить информацию из некоторой ячейки основной памяти, а копия содержимого этой ячейки уже имеется в кэш-памяти (первая строка табл. 3.1), то вместо оригинала считывает-

ся копия. Информация в кэш-памяти и основной памяти не изменяется. Если копии нет, то производится обращение к основной памяти. Полученная информация пересылается в процессор и попутно запоминается в кэш-памяти. Чтение информации в отсутствие копии отражено во второй строке таблицы. Информация в основной памяти не изменяется.

При записи существует несколько методов обновления старой информации. Эти методы называются **стратегией обновления строк основной памяти**. Если результат обновления строк кэш-памяти не возвращается в основную память, то содержимое основной памяти становится неадекватным вычислительному процессу. Чтобы избежать этого, предусмотрены методы обновления основной памяти, которые можно разделить на две большие группы: метод сквозной записи и метод обратной записи.

Таблица 3.1Условия сохранения и обновления информации

Режим работы	Наличие копии ячейки ОП в кэш-памяти	Информация	
		В ячейке кэш-памяти	В ячейке основной памяти
Чтение	Копия есть. Копии нет	Не изменяется. Обновляется (создается копия)	Не изменяется. Не изменяется
Сквозная запись	Копия есть. Копии нет	Обновляется. Не изменяется	Обновляется. Обновляется
Обратная запись	Копия есть. Копии нет	Обновляется. Создается копия. Обновляется	Не изменяется. Не изменяется

Сквозная запись

По методу сквозной записи обычно обновляется слово, хранящееся в основной памяти. Если в кэш-памяти существует копия этого слова, то она также обновляется. Если же в кэш-памяти отсутствует копия этого слова, то либо из основной памяти в кэш-память пересылается строка, содержащая это слово (метод WTWA – сквозная запись с распределением), либо этого не допускается (метод WTNWA – сквозная запись без распределения). Когда по методу сквозной записи область (строка) в кэш-памяти назначается для хранения другой строки, то в основную память можно не возвращать удаляемый блок, т.к копия там есть. Однако в этом случае эффект от использования кэш-памяти отсутствует.

Обратная запись

По методу обратной записи, если адрес объектов, по которым есть запрос обновления, существует в кэш-памяти, то обновляется только

кэш-память, а основная память не обновляется. Если адреса объекта обновления нет в кэш-памяти, то в неё из основной памяти пересылается строка, содержащая этот адрес, после чего обновляется только кэш-память. По методу обратной записи, в случае замены строк, удаляемую строку необходимо также пересылать в основную память. У этого метода существуют две разновидности: метод SWB (простая обратная запись), по которому удаляемая строка возвращается в основную память, и метод FWB (флаговая обратная запись), по которому в основную память записывается только обновлённая строка кэш-памяти. В последнем случае каждая область строки в кэш-памяти снабжается однобитовым флагом, который показывает, было или нет обновление строки, хранящейся в кэш-памяти. Метод FWB обладает достаточной эффективностью, однако более эффективным считается метод FPWB (флаговая регистровая обратная запись), в котором, благодаря размещению буфера между кэш-памятью и основной памятью, предотвращается конфликт между удалением и выборкой строк.

Таким образом, теоретически более предпочтительным алгоритмом записи для кэша является метод обратной записи. Кэш с обратной записью будет хранить новую информацию до тех пор, пока у него не появится необходимость избавиться от неё. Тем самым процессор может более оперативно управлять системой. В связи с тем, что кэш со сквозной записью сразу же передаёт вновь записанную информацию в память следующего уровня, кэш со сквозной записью может вызывать дополнительные потери в быстродействии по сравнению с кэшем с обратной записью. В случае кэша с обратной записью допускается выполнение длинных последовательностей быстрых операций записи из процессора, поскольку нет необходимости немедленно направлять эти данные в основную память.

3.3.4 Методы замещения строк кэш-памяти

Способ определения строки, удаляемой из кэш-памяти, называется **стратегией замещения**. Для замещения строк кэш-памяти существует несколько методов:

- замещение строки, к которой наиболее длительное время не было обращения (метод LRU);
- замещение строки, загруженной в кэш-память первой (метод FIFO);
- произвольное замещение.

Реализация этих методов упрощается в указанной последовательности, но наибольшим эффектом обладает метод замещения наиболее давнего по использованию объекта (строки).

Для реализации этого метода необходимо манипулировать строками, которые являются объектами замещения, с помощью LRU-стека. При каждой загрузке в этот стек помещается строка, в результате чего при замене используется строка, хранящаяся в наиболее глубокой позиции стека, и эта строка удаляется из стека. При доступе к строке, которая уже содержится в LRU-стеке, эта строка удаляется из стека и заново загружается в него. Стек типа LRU устроен таким образом, что чем дольше к строке не было доступа, тем в более глубокой позиции она располагается. Реализация стека типа LRU, позволяющего с высокой скоростью выполнять такую операцию, усложняется по мере увеличения числа строк.

3.3.5 Многоуровневая организация кэша

Предельно достижимая ёмкость кэш-памяти ограничена не только её ценой, но и электромагнитной интерференцией, налагающей жёсткие ограничения на максимально возможное количество адресных линий, а значит – на непосредственно адресуемый объём памяти. В принципе, можно прибегнуть к мультиплексированию выводов или последовательной передаче адресов, но это неизбежно снизит производительность и увеличит время доступа к ячейке кэш-памяти. С другой стороны, двухпортовая статическая память действительно очень дорогая, а однопортовая не в состоянии обеспечить параллельную обработку нескольких ячеек, что приводит к досадным задержкам. Естественный выход состоит в создании многоуровневой кэш-иерархии (рис. 3.7).

Большинство современных компьютеров имеют два или три уровня кэш-памяти. Первый, наиболее «близкий» к ядру процессора (L1), обычно реализуется на быстрой двухпортовой синхронной статической памяти, работающей на полной частоте ядра. Объём L1 кэша весьма невелик, составляет 64 Кб или 128 Кб и разделяется пополам на два кэша данных и команд для каждого ядра процессора. Латентность кэша L1 измеряется 3-мя, 4-мя тактами. На втором уровне расположен кэш L2. Он реализуется на однопортовой конвейерной статической памяти и зачастую работает на пониженной тактовой частоте. Поскольку однопортовая память значительно дешевле, объём L2 кэша достигает нескольких мегабайт в двухъядерных структурах процессоров, когда он является общим для двух ядер (Intel Core 2 Duo), или несколько сотен кило-

байт (256 Кб или 512 Кб), когда в многоядерном процессоре каждое ядро имеет свой L2 кэш (рис. 3.10). Этот кэш хранит как команды, так и данные. Латентность L2 для процессоров Intel Nehalem 3,2 ГГц составляет 11 тактов, для Penryn 3,2 ГГц – 18 тактов.

На третьем уровне находится L3 кэш, который объединяет ядра между собой и является разделяемым. В результате L2 кэш выступает в качестве буфера при обращениях процессорных ядер в разделяемую кэш-память, имеющую достаточно солидный объём (2 Мб – AMD K10, 8 Мб – Intel Nehalem). Латентность L3 кэша исчисляется 52-мя, 54-мя тактами.

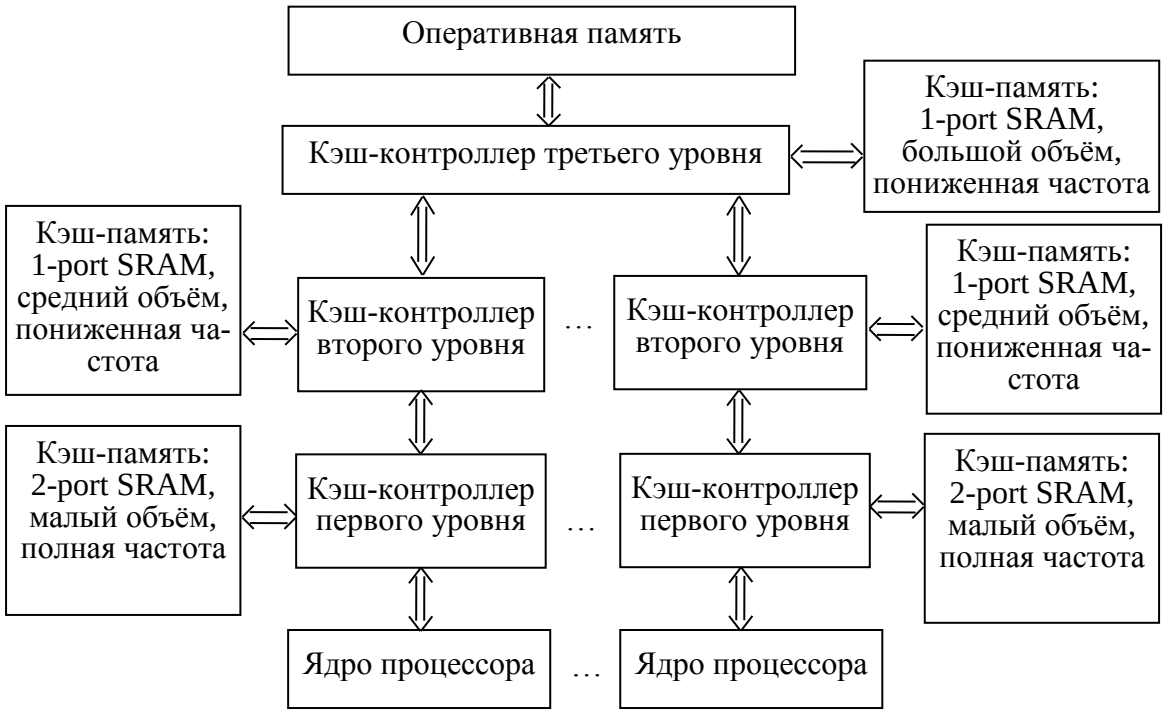


Рис. 3.10. Трехуровневая структура кэш-памяти многоядерного процессора

При построении многоуровневой кэш-памяти используют **включающую** (inclusive) или **исключающую** (exclusive) технологии. Кэш верхнего уровня, построенный по **inclusive-технологии**, всегда дублирует содержимое кэша нижнего уровня. Если построить инклюзивный L3 кэш, то он будет дублировать данные, хранящиеся в кэшах первого и второго уровней, что снижает эффективную ёмкость всей кэш-подсистемы. С другой стороны, инклюзивный разделяемый L3 кэш способен обеспечить в многоядерных процессорах более высокую скорость работы подсистемы памяти. Это связано с тем, что если ядро попытается получить доступ к данным и они отсутствуют в кэше L3, то нет необ-

ходимости искать эти данные в собственных кэшах других ядер – там их нет. А благодаря тому, что каждая строка L3 кэша снабжена дополнительными флагами, указывающими владельцев (ядра) этих данных, не вызывает затруднений и процедура обратного изменения содержимого строки кэша. Так, если какое-то ядро модифицирует данные в L3 кэше, изначально принадлежащие другому (или другим) ядру, то в этом случае обновляется содержимое L1 и L2 кэшей и этих ядер. Эта технология весьма эффективна для обеспечения когерентности персональных кэшей каждого ядра, поскольку она уменьшает потребность в обмене информацией между ядрами. По такой технологии организована кэш-память процессоров Intel Nehalem.

Кэш – подсистема, построенная по **exclusive-технологии**, никогда не хранит избыточных копий данных и потому эффективная ёмкость подсистемы определяется суммой ёмкостей кэш-памятей всех уровней. Кэш первого уровня никогда не уничтожает строки при нехватке места. Даже если они не были модифицированы, данные в обязательном порядке вытесняются в кэш второго уровня, помещаясь на то место, где находилась только что переданная кэшу L1 строка, т.е. кэши L1 и L2 как бы обмениваются друг с другом своими строками, а потому кэш-память используется весьма эффективно. По такой технологии организована кэш-память процессоров AMD K10.

Для определения среднего времени доступа процессора к данным с учетом кэш-поданий и промахов разработчиками используется значение, называемое AMAT (Averag Memory Access Time), которое расчитывается по формуле:

AMAT = время попадания + коэффициент промахов * издержка промаха.

Под издержкой промаха подразумевается время обращения к **оперативной** памяти.

3.4 Принципы организации оперативной памяти

3.4.1 Общие положения

Оперативная (основная) память представляет собой следующий уровень иерархии памяти. Оперативная память удовлетворяет запросы кэш-памяти и устройств ввода/вывода. Она является местом назначения для ввода и источником для вывода. Для оценки производительности (быстродействия) основной памяти используются следующие параметры: **время доступа, длительность цикла памяти, латентность и пропускная способность.**

Как было сказано выше, **время доступа** – это время, проходящее с момента обращения к памяти до момента считывания данных. Данная величина приблизительно одинакова для всех типов динамической памяти и составляет примерно 50 нс. Время доступа актуально при случайном доступе к памяти, т.е. когда последовательно считываемые ячейки памяти принадлежат различным строкам матрицы памяти.

Если говорить о блочной передаче, то более показательной характеристикой является **время цикла,** т.е. время между двумя последовательными обращениями к ячейкам памяти. Первый цикл обращения всегда равен времени доступа, т.е. около 50 нс. Но при последующих циклах обращения в пределах одной страницы (строки матрицы) время существенно меньше и составляет 10 или 7,5 нс. Любая динамическая память характеризуется циклами доступа, записываемыми в виде цепочек типа 5–1–1–1 или 5–2–2–2 и т.д. Такая цепочка определяет количество тактов, необходимых для чтения первых четырех элементов (байт, слово, двойное слово) данных в страничном режиме доступа. Первая цифра в таком обозначении определяет время доступа, т.е. количество тактов, прошедших от начала обращения к банку памяти до появления данных на шине. Соответственно, при работе в страничном режиме следующие данные появятся на шине уже через меньшее количество тактов. Например, при цепочке 5–1–1–1 последующие данные появляются без задержек, т.е. с каждым тактовым импульсом.

Латентность памяти определяется некоторым набором значений временных задержек, происходящих в модуле памяти с момента прихода команды чтения (записи) до ее выполнения. Эти значения задержек принято называть **таймингами.** При описании памяти принято использовать четыре тайминга – *t*CL, *t*RCD, *t*RP, *t*RAS (иногда дополнительно указывается и Command rate), причем записываются они обычно в этой же последовательности в виде 4–4–4–12 (1T), где цифры указывают количество затраченных тактов синхронизации (в данном случае цифровые значения взяты произвольно).

Перед тем как расшифровать аббревиатуры указанных таймингов, несколько слов о принципах организации и работы оперативной памяти. Ядро памяти организовано в виде двумерной матрицы. Для получения доступа к той или иной ячейке необходимо указать адреса соответствующей строки и столбца. Для ввода адреса строки используется стробирующий сигнал RAS, а для адреса столбца – стробирующий сигнал CAS. Порядок обращения к памяти начинается с установки регистров управления, после чего вырабатывается сигнал выбора нужного банка памяти и по прошествии (задержки) Command rate осуществляется ввод

адреса строки и подача стробирующего сигнала RAS (обычно эта задержка составляет один или два такта). С приходом положительного фронта тактового импульса открывается доступ к нужной строке, а адрес строки помещается в адресный буфер строки, где он может удерживаться столько времени, сколько нужно. Через промежуток времени, называемый RAS to CAS delay (*t*RCD), т.е. задержка подачи сигнала CAS относительно сигнала RAS, подается стробирующий импульс CAS, под действием которого происходит выборка адреса столбца и открывается доступ к нужному столбцу матрицы памяти. Затем, через время CAS latency (tCL), на шине данных появляется первое слово, которое может быть считано процессором. После завершения работы со всеми ячейками активной строки выполняется команда деактивации Precharge, позволяющая перейти к следующей строке (*t*RP – это time of Row Precharge: тайминг между завершением обработки одной строки и переходом к другой). Значение *t*RAS (time of Active to Precharge Delay) считается одним из основных параметров, поскольку он описывает время задержки между моментом активации строки и моментом подачи команды деактивации Precharge, которой заканчивается работа с этой строкой. Общее правило гласит: чем меньше тайминги при одной тактовой частоте, тем быстрее память. Более того, в целом ряде случаев быстрее оказывается память с меньшими таймингами, работающая даже на более низкой тактовой частоте. Память с более высокой тактовой частотой имеет, как правило, более высокие тайминги.

Другой важнейшей характеристикой ОП является ее **пропускная способность**, которая определяется как произведение частоты работы памяти на объем данных, передаваемых за один такт. Самый простой способ увеличения максимальной пропускной способности памяти заключается в увеличении частоты ее работы. Однако на практике реализовать это совсем не просто. Вспомним, что элементарной ячейкой динамической памяти является конденсатор – инерционное по своей природе устройство. Чтобы произвести считывание информации с конденсатора, необходимо его разрядить, для чего требуется определенное время, пропорциональное емкости конденсатора, – сделать это мгновенно невозможно. Следовательно, нельзя повышать частоту ядра памяти до бесконечности. Кроме того, динамическая память требует периодической регенерации, чтобы восстанавливать заряды конденсаторов, а для зарядки конденсаторов тоже необходим определенный временной интервал. В результате повышение частоты ядра памяти сопряжено с непреодолимыми трудностями. Конечно, применение более миниатюрных конденсаторов повышает их быстродействие, однако для этого нужно использовать иную проектную норму при производстве чипов

памяти. К тому же переход на новый технологический процесс производства не может кардинально увеличить скорость работы памяти. Поэтому, кроме банального увеличения частоты работы памяти, для увеличения ее пропускной способности часто используют другие приемы.

3.4.2 Методы повышения пропускной способности ОП

Согласование производительности современных процессоров со скоростью ОП остается одной из важнейших проблем. Методы повышения производительности за счет увеличения размеров кэш-памяти и введения многоуровневой организации кэш-памяти полностью не решают эту проблему. Поэтому важным направлением современных разработок являются методы повышения пропускной способности памяти за счет ее организации, включая специальные способы организации DRAM.

Развитие способов организации памяти DDR SDRAM

Кардинальным способом увеличения пропускной способности ОП стал переход к стандарту DDR. Динамическая память DDR SDRAM пришла на смену синхронной SDRAM и обеспечила в два раза большую пропускную способность. Аббревиатура DDR (Double Data Rate) означает удвоенную скорость передачи данных. Как уже отмечалось выше, основным сдерживающим элементом увеличения тактовой частоты работы памяти является ядро памяти (массив элементов хранения – Memory Cell Array). Однако, кроме ядра, в модуле памяти присутствуют и буферы промежуточного хранения (буферы ввода/вывода – I/O Buffers), через которые ядро памяти обменивается данными с шиной памяти. Эти буферы могут иметь значительно более высокое быстродействие, чем само ядро, поэтому тактовую частоту работы шины памяти и буферов обмена можно легко увеличить. Именно такой способ и используется в DDR-памяти.

Рассмотрим предельно упрощенную схему функционирования памяти типа SDRAM (рис. 3.11, *а*). Ядро SDRAM-памяти и буферы ввода/вывода работают в синхронном режиме на одной и той же частоте. Передача каждого бита из буфера на шину происходит с каждым тактом работы ядра памяти.

При переходе от SDRAM к DDR (рис. 3.11, *б*) технология одинарной скорости передачи данных заменяется на удвоенную за счет того, что передача данных от микросхем памяти модуля к контроллеру памяти по внешней шине данных осуществляется по обоим полупериодам синхросигнала (восходящему – «фронту», и нисходящему – «срезу»). В

этом и заключается суть технологии «Double Data Rate – DDR», именно поэтому «эффективная» частота памяти DDR-400 составляет 400 МГц, тогда как ее истинная частота, или частота буферов ввода/вывода, составляет 200 МГц. Таким образом, каждый буфер ввода-вывода передает на шину два бита информации за один такт, оставаясь при этом полностью синхронизированным с ядром памяти. Однако, чтобы такой режим работы стал возможным, необходимо, чтобы эти два бита были доступны буферу ввода/вывода на каждом такте работы памяти. Для этого требуется, чтобы каждая команда чтения приводила к передаче из ядра памяти в буфер сразу двух бит по двум независимым линиям передачи внутренней шины данных. Из буфера ввода/вывода биты данных затем поступают на внешнюю шину в требуемом порядке. Иными словами, можно сказать, что, при прочих равных условиях, внутренняя шина данных должна быть вдвое шире по сравнению с внешней шиной данных. Такая схема доступа к данным называется схемой «2n-предвыборки» (2n-prefetch). DDR-память, как и SDRAM, предназначалась для работы с системными частотами 100, 133, 166, 200, 216, 250 и 266 МГц. Нетрудно рассчитать пропускную способность DDR-памяти. Принимая, что ширина внешней шины данных составляет 8 байт, для памяти
DDR-400 получаем 400 МГц × 8 байт = 3,2 Гбайт/с.

Наиболее естественным путем решения проблемы достижения более высоких тактовых частот при переходе от DDR к DDR2 явилось снижение тактовой частоты внутренней шины данных вдвое по отношению к реальной тактовой частоте внешней шины данных (частоте буферов ввода/вывода). Так, в рассматриваемом примере микросхем памяти DDR2-800 (рис. 3.11, *в*) частота буферов составляет 400 МГц, а «эффективная» частота внешней шины данных – 800 МГц (поскольку сущность технологии Double Data Rate остается в силе). При этом частота внутренней шины данных (ядра памяти) составляет всего 200 МГц, поэтому для передачи 1 бита (по каждой линии данных) за такт внешней шины с «эффективной» частотой 800 МГц на каждом такте внутренней шины данных требуется передача уже 4 бит данных. Иными словами, внутренняя шина данных микросхемы памяти DDR2 должна быть в 4 раза шире по сравнению с её внешней шиной. Такая схема доступа к данным называется схемой «4n-предвыборки» (4n-prefetch). Ее преимущества перед схемой 2n-prefetch, реализованной в DDR, очевидны. С одной стороны, для достижения равной пиковой пропускной способности можно использовать вдвое меньшую внутреннюю частоту микросхем памяти (200 МГц для DDR-400 и всего 100 МГц для DDR2-

400), что позволяет значительно снизить энергопотребление. С другой стороны, при равной внутренней частоте функционирования микросхем DDR и DDR2 (200 МГц как для DDR-400, так и DDR2-800) последние будут характеризоваться вдвое большей теоретической пропускной способностью. Но очевидны и недостатки: функционирование буферов ввода/вывода микросхем DDR2 на вдвое большей частоте и использование более сложной схемы преобразования «4–1» приводит к ощутимому возрастанию задержек (таймингов).

Очередной «эволюционный скачок» в технологии реализации памяти DDR SDRAM – это переход от стандарта DDR2 к новому стандарту DDR3. Нетрудно догадаться, что основной принцип, лежащий в основе перехода от DDR2 к DDR3, в точности повторяет рассмотренную выше идею, заложенную при переходе от DDR к DDR2.

А именно, DDR3 – это всё та же удвоенная частота внешней шины данных по отношению к частоте внутренней шины, это удвоенная частота буферов ввода/вывода по сравнению с DDR2. Типичными скоростными категориями памяти нового стандарта DDR3 являются разновидности DDR3-800, DDR3-1066, DDR3-1333, DDR3-1600, DDR3-1866. Очередное увеличение теоретической пропускной способности компонентов памяти в 2 раза вновь связано со снижением их внутренней частоты функционирования во столько же раз. Поэтому для достижения темпа передачи данных со скоростью 1 бит/такт по каждой линии внешней шины данных с «эффективной» частотой в 1600 МГц (как в примере, рассмотренном на рис. 3.11, *г*) используемые микросхемы (с частотой 200 МГц) должны передавать по 8 бит данных за каждый «свой» такт, т.е. ширина внутренней шины данных микросхем памяти окажется уже в 8 раз больше по сравнению с шириной их внешней шины. Такая схема передачи данных с рассмотренным преобразованием типа «8–1» называется схемой «8n-предвыборки» (8n-prefetch).

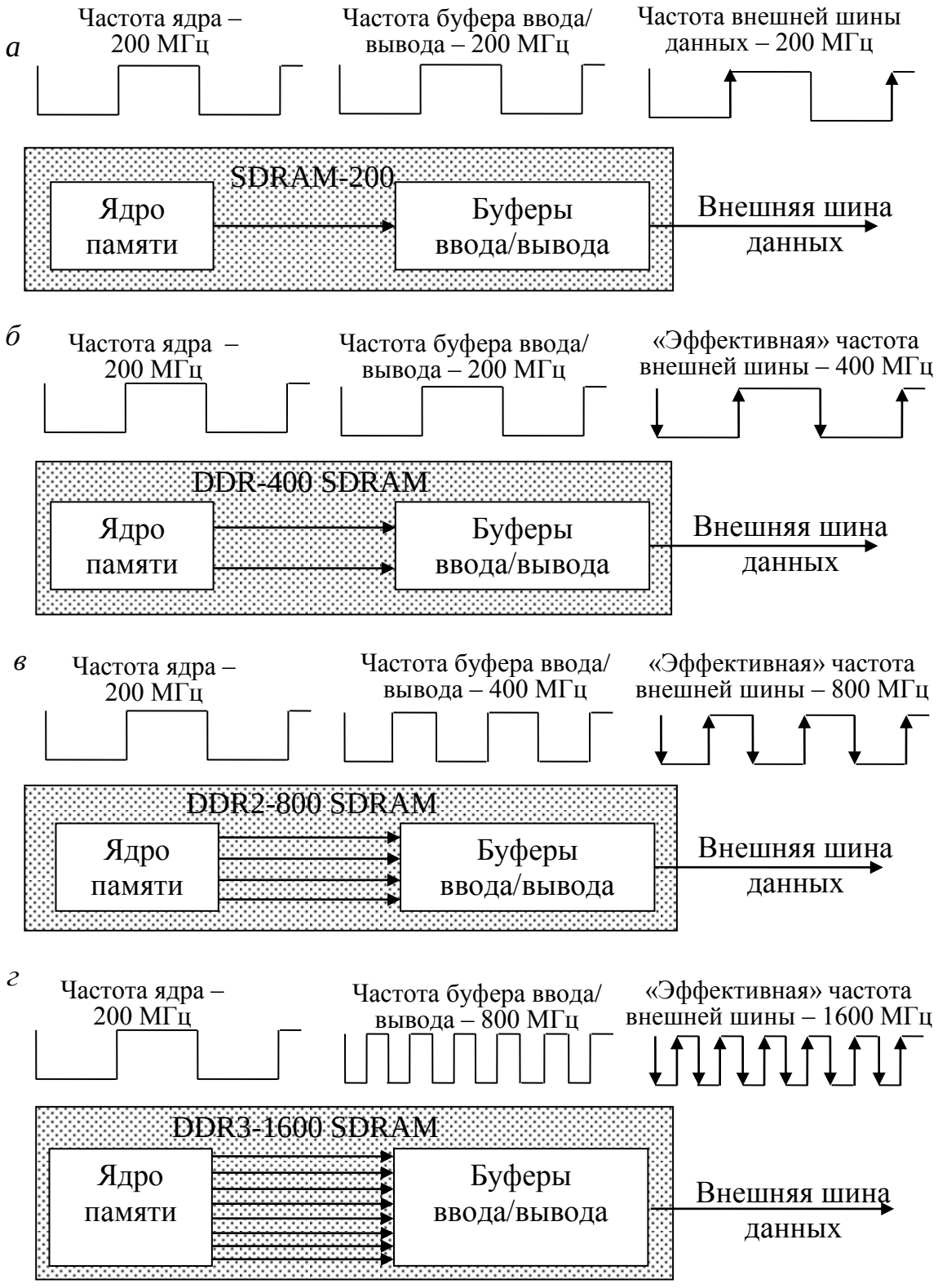


Рис. 3.11. Схематическое представление передачи данных в микросхеме памяти: *а* – SDRAM-200; *б* – DDR-400; *в* – DDR2-800; *г* – DDR3-1600

Преимущества при переходе от DDR2 к DDR3 те же, что и при состоявшемся ранее переходе от DDR к DDR2: с одной стороны, это снижение энергопотребления компонентов в условиях равенства их пиковой пропускной способности (DDR3-800 против DDR2-800), с другой стороны – возможность дальнейшего наращивания тактовой частоты и теоретической пропускной способности при сохранении прежнего уровня «внутренней» частоты компонентов (DDR3-1600 против DDR2-800). Теми же являются и недостатки: дальнейший разрыв между «внутренней» и «внешней» частотами шин компонентов памяти приводит к ещё большим задержкам.

Дальнейшее развитие технологии реализации памяти DDR SDRAM – это переход к новому стандарту DDR4. По сравнению с DDR3 эта память работает на более высоких частотах (DDR4 – 3200, DDR4 – 3000, DDR4 – 2800, DDR4 – 2666), благодаря чему она может обеспечить гораздо лучшую пропускную способность, однако при этом её латентность заметно выше.

Чтобы проиллюстрировать вышесказанное приведем (рис. 3.12) с помощью бенчмарков из пакета SiSoftware Sandra 21.42 практические характеристики подсистемы памяти Skylake-S, укомплектованной модулями DDR4 SDRAM с различной скоростью. Для сравнения рядом приводятся результаты, полученные в аналогичной системе с процессором Haswell и привычной памятью типа DDR3 SDRAM.

Как нетрудно заметить, DDR4 SDRAM действительно обеспечивает более высокую пропускную способность. Она продолжает масштабироваться синхронно с частотой, то есть модули DDR4 с более высокой задекларированной скоростью всегда смогут обеспечить лучшую полосу пропускания по сравнению с DDR3. И это – несомненное преимущество новой технологии. Однако латентность серьёзно повысилась. Как видно из диаграмм, такие же, как у DDR3-1866, задержки можно получить лишь при установке в систему модулей класса DDR4-3000, которые относятся к числу премиальных оверклокерских предложений. Иными словами, переход с DDR3 на DDR4 пока не несёт очевидных плюсов. В каких-то аспектах быстродействия системы нового поколения от использования иной технологии памяти выиграют, но в каких-то и проиграют.

Правда, не стоит упускать из виду, что с выходом Skylake внедрение DDR4 в массовых системах должно заметно ускорить продвижение этой технологии. Так что недорогие и скоростные модули DDR4 SDRAM, которые будут превосходить память предшествующего стан-

дарта во всех аспектах, могут приобрести широкое распространение уже в самом ближайшем будущем.

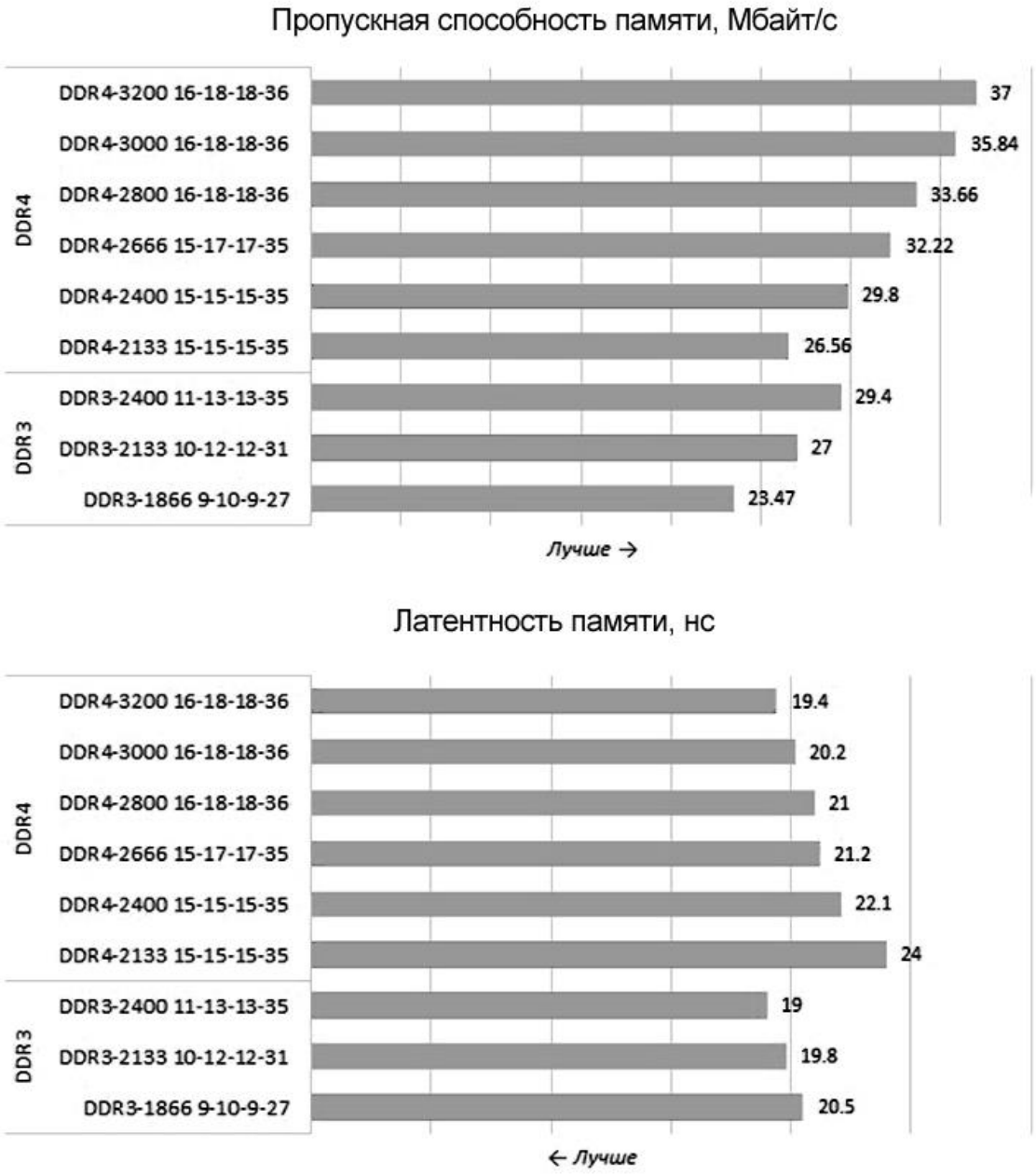


Рис. 3.12. Характеристики памяти на DDR4 в сравнении с DDR3

Выборка широким словом

Прямой способ сокращения числа обращений к ОП состоит в организации **выборки широким словом.** Этот способ основывается на свойстве локальности данных и программ. При выборке широким словом за одно обращение к ОП производится одновременная запись или считывание нескольких команд или слов данных из «широкой» ячейки. Широкое слово заносится в буферную память (кэш-память) или регистр, где оно расформировывается на отдельные команды или слова данных, которые могут последовательно использоваться процессором без дополнительных обращений к ОП.

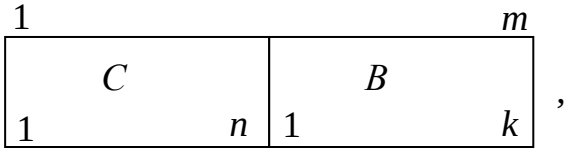
В системах с кэш-памятью первого уровня ширина шин данных ОП часто соответствует ширине шин данных кэш-памяти, которая во многих случаях имеет физическую ширину шин данных, соответствующую количеству разрядов в слове. Удвоение и учетверение ширины шин кэш-памяти и ОП удваивает или учетверяет соответственно полосу пропускания системы памяти.

Реализация выборки широким словом вызывает необходимость мультиплексирования данных между кэш-памятью и процессором, поскольку основной единицей обработки данных в процессоре все еще остается слово. Эти мультиплексоры оказываются на критическом пути поступления информации в процессор. Кэш-память второго уровня несколько смягчает эту проблему, в этом случае мультиплексоры могут располагаться между двумя уровнями кэш-памяти, т.е. вносимая ими задержка не столь критична. Другая проблема, связанная с увеличением разрядности памяти, заключается в необходимости определения минимального объема (инкремента) памяти для поэтапного её расширения, которое часто выполняется самими пользователями во время эксплуатации системы. Удвоение или учетверение ширины памяти приводит к удвоению или учетверению этого минимального инкремента. Кроме того, имеются проблемы и с организацией коррекции ошибок в системах с широкой памятью.

Расслоение обращений

Другой способ повышения пропускной способности ОП связан с построением памяти, состоящей на физическом уровне из нескольких модулей (банков) с автономными схемами адресации, записи и чтения. При этом на логическом уровне управления памятью организуются последовательные обращения к различным физическим модулям. Обращения к различным модулям могут перекрываться, и таким образом образуется своеобразный конвейер. Эта процедура носит название **рас-**

слоения памяти. Целью данного метода является увеличение скорости доступа к памяти посредством совмещения фаз обращений ко многим модулям памяти. Известно несколько вариантов организации расслоения. Наиболее часто используется способ расслоения обращений за счет **расслоения адресов.** Этот способ основывается на свойстве локальности программ и данных, предполагающем, что адрес следующей команды программы на единицу больше адреса предыдущей (линейность программ нарушается только командами перехода). Аналогичная последовательность адресов генерируется процессором при чтении и записи слов данных. Таким образом, типичным случаем распределения адресов обращений к памяти является последовательность вида a, $a + 1$, $a + 2$, … Из этого следует, что расслоение обращений возможно, если ячейки с адресами a, $a + 1$, $a + 2$, … будут размещаться в блоках 0, 1, 2, … Такое распределение ячеек по модулям (банкам) обеспечивается за счет использования адресов вида



,

где B – k-разрядный адрес модуля (младшая часть адреса) и C – n-разрядный адрес ячейки в модуле B (старшая часть адреса).

Принцип расслоения обращений иллюстрируется на рис. 3.12, *а*. Все программы и данные «размещаются» в адресном пространстве последовательно. Однако ячейки памяти, имеющие смежные адреса, находятся в различных физических модулях памяти. Если ОП состоит из 4-х модулей, то номер модуля кодируется двумя младшими разрядами адреса. При этом полные m-разрядные адреса 0, 4, 8, … будут относиться к блоку 0, адреса 1, 5, 9, … – к блоку 1, адреса 2, 6, 10, … – к блоку 2 и адреса 3, 7, 11, … – к блоку 3. В результате этого последовательность обращений к адресам 0, 1, 2, 3, 4, 5, … будет расслоена между модулями 0, 1, 2, 3, 0, 1, ….

Поскольку каждый физический модуль памяти имеет собственные схемы управления выборкой, можно обращение к следующему модулю производить, не дожидаясь ответа от предыдущего. Так на временной диаграмме (рис. 3.12, *б*) показано, что время доступа к каждому модулю составляет $\tau = 4T$, где $T = t_{i+1} - t_i$ – длительность такта. В каждом такте следуют непрерывно обращения к модулям памяти в моменты времени t_1, t_2, t_3, ...

При наличии четырех модулей темп выдачи квантов информации из памяти в процессор будет соответствовать одному такту T, при этом скорость выдачи информации из каждого модуля в четыре раза ниже.

Задержка в выдаче кванта информации относительно момента обращения также составляет $4T$, однако задержка в выдаче каждого последующего кванта относительно момента выдачи предыдущего составит T.

При реализации расслоения по адресам число модулей памяти может быть произвольным и необязательно кратным степени 2. В некоторых компьютерах допускается произвольное отключение модулей памяти, что позволяет исключать из конфигурации неисправные модули.

В современных высокопроизводительных компьютерах число модулей обычно составляет 4–16, но иногда превышает 64.

Так как схема расслоения по адресам базируется на допущении о локальности, она дает эффект в тех случаях, когда это допущение справедливо, т.е. при решении одной задачи.

Для повышения производительности мультипроцессорных систем, работающих в многозадачных режимах, реализуют другие схемы, при которых **различные процессоры обращаются к различным модулям** памяти. Необходимо помнить, что процессоры ввода/вывода также занимают циклы памяти и вследствие этого могут сильно влиять на производительность системы. Для уменьшения этого влияния обращения центрального процессора и процессоров ввода/вывода можно организовать к разным модулям памяти.

Обобщением идеи расслоения памяти является **возможность реализации нескольких независимых обращений**, когда несколько контроллеров памяти позволяют модулям памяти (или группам расслоенных модулей памяти) работать независимо.

Прямое уменьшение числа конфликтов за счет организации чередующихся обращений к различным модулям памяти может быть достигнуто путем **размещения программ и данных в разных модулях.** Доля команд в программе, требующих ссылок к находящимся в ОП данным, зависит от класса решаемой задачи и от архитектурных особенностей компьютера. Для большинства ЭВМ с традиционной архитектурой и задач научно-технического характера эта доля превышает 50 %.

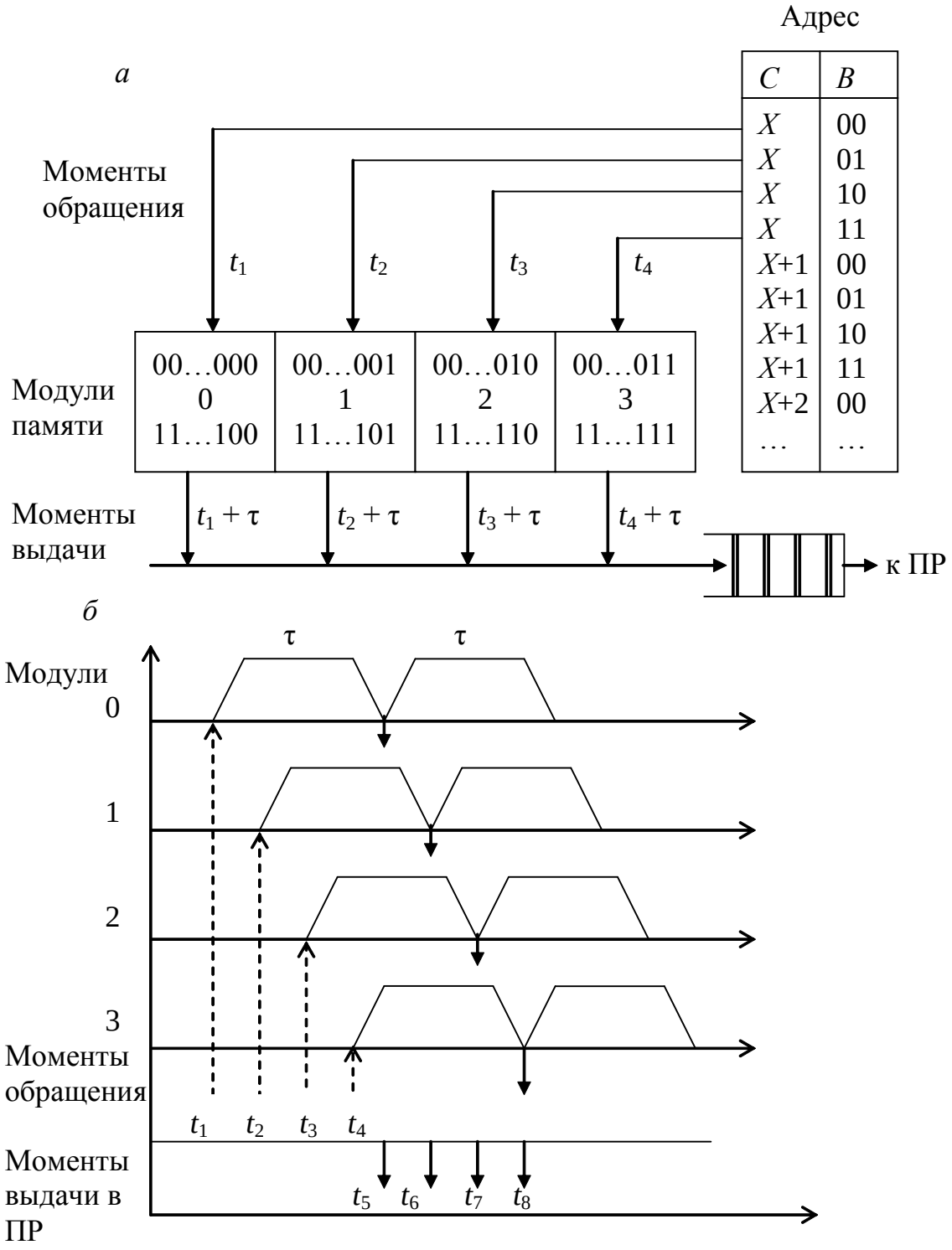


Рис. 3.12. Расслоение памяти:
а – организация адресного пространства; *б* – временная диаграмма работы модулей

Поскольку обращения к командам и элементам данных чередуются, то разделение памяти на память команд и память данных повышает быстродействие машины подобно рассмотренному выше механизму расслоения. Разделение памяти на память команд и память данных широко используется в системах управления или обработки сигналов. В подобного рода системах в качестве памяти команд нередко используются постоянные запоминающие устройства (ПЗУ), цикл которых меньше цикла устройств, допускающих запись, это делает разделение программ и данных весьма эффективным. Следует отметить, что обращения процессоров ввода/вывода в режиме прямого доступа в память логически реализуются как обращения к памяти данных.

Выбор той или иной схемы расслоения для компьютера (системы) определяется целями (достижение высокой производительности при решении множества задач или высокого быстродействия при решении одной задачи), архитектурными и структурными особенностями системы, а также элементной базой (соотношением длительностей циклов памяти и узлов обработки). Могут использоваться комбинированные схемы расслоения.

3.4.3 Методы управления памятью

Оперативная память является важнейшим и наиболее дефицитным ресурсом в вычислительных машинах и системах, требующим тщательного и эффективного управления. Проблема усложняется при переходе к мультипрограммным системам, т.к в них оперативную память одновременно используют несколько вычислительных процессов (программ).

Типы адресов

Для идентификации переменных и команд используются символьные имена (метки), виртуальные адреса и физические адреса (рис. 3.13).

Символьные имена присваивает пользователь при написании программы на алгоритмическом языке или ассемблере.

Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык. Так как во время трансляции в общем случае неизвестно, в какое место ОП будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, обычно считая по умолчанию, что программа будет размещена, начиная с нулевого адреса. Совокупность виртуальных адресов процесса (программы) называется виртуальным адресным пространством. Каждый процесс имеет собственное виртуальное адресное простран-

ство. Максимальный размер виртуального адресного пространства ограничивается разрядностью адреса, присущей данной архитектуре компьютера и, как правило, не совпадает с объемом физической памяти, имеющимся в компьютере.

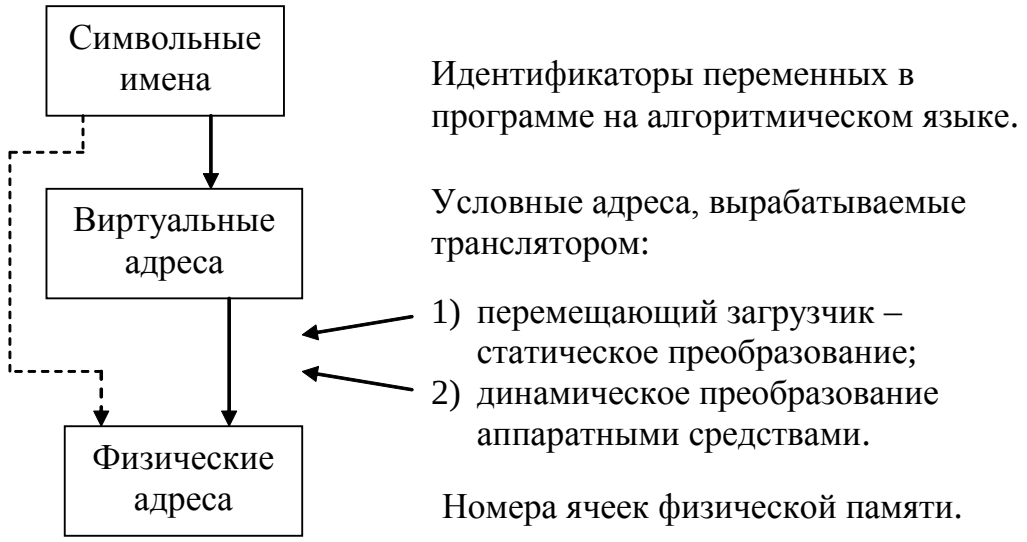


Рис. 3.13. Типы адресов

Физические адреса соответствуют номерам ячеек ОП, где в действительности расположены или будут расположены переменные и команды. Переход от виртуальных адресов к физическим может осуществляться двумя способами.

В первом случае замену виртуальных адресов на физические делает специальная системная программа – **перемещающий загрузчик**. Перемещающий загрузчик на основании имеющихся у него исходных данных о начальном адресе физической памяти, в которую предстоит загружать программу, и информации, предоставленной транслятором об адресно-зависимых константах программы, выполняет загрузку программы, совмещая её с заменой виртуальных адресов физическими.

Второй способ заключается в том, что программа загружается в память в неизменном виде в виртуальных адресах, при этом операционная система (ОС) фиксирует смещение действительного расположения программного кода относительно виртуального адресного пространства. Во время выполнения программы при каждом обращении к ОП выполняется **преобразование виртуального адреса в физический**. Второй способ является более гибким, он допускает перемещение программы во время ее выполнения, в то время как перемещающий загруз-

чик жестко привязывает программу к первоначально выделенному ей участку. Вместе с тем использование загрузчика уменьшает накладные расходы, т.к преобразование каждого виртуального адреса происходит только один раз во время загрузки, а во втором случае – каждый раз при обращении по данному адресу.

В некоторых случаях (обычно в специализированных системах), когда заранее точно известно, в какой области ОП будет выполняться программа, транслятор выдает исполняемый код сразу в физических адресах.

Классификация методов распределения оперативной памяти

Все методы управления памятью могут быть разделены на два класса (рис. 3.14):

- методы распределения ОП без использования внешней памяти (дискового пространства);
- методы распределения памяти с использованием дискового пространства.

Первая группа включает методы распределения памяти фиксированными, динамическими и перемещаемыми разделами. Вторая – страничное, сегментное и странично-сегментное распределение памяти.

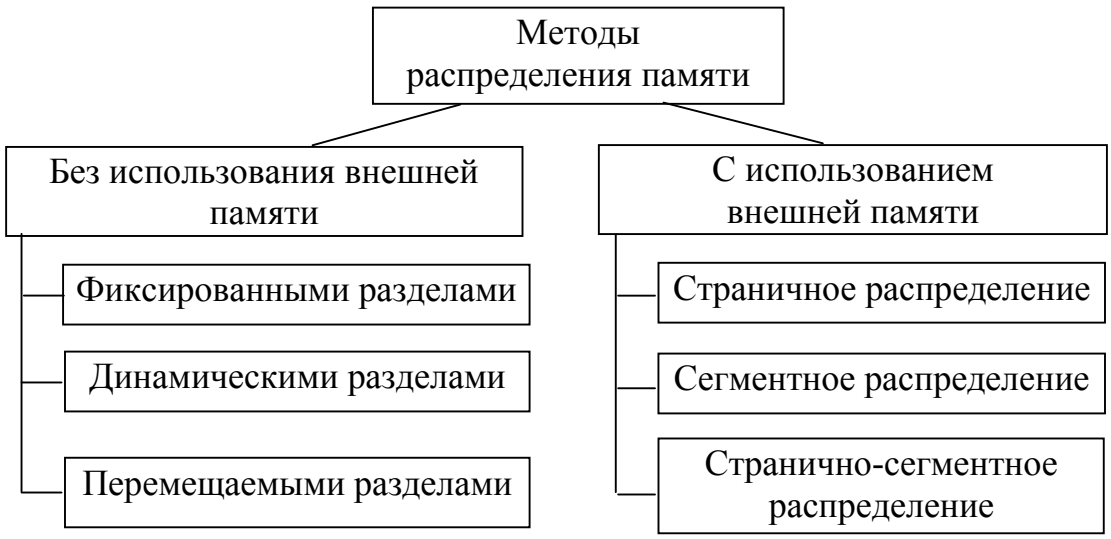


Рис. 3.14. Классификация методов распределения ОП

Рассмотрим вначале первую группу методов.

Распределение памяти фиксированными разделами

Самым простым способом управления оперативной памятью является разделение её на несколько разделов (сегментов) фиксированной величины (**статическое распределение**). Это может быть выполнено вручную оператором во время старта системы или во время её генерации. Очередная задача, поступающая на выполнение, помещается либо в общую очередь (рис. 3.15, *а*), либо в очередь к некоторому разделу (рис. 3.15, *б*). Подсистема управления памятью в этом случае выполняет следующие задачи: сравнивает размер программы, поступившей на выполнение, и свободных разделов памяти; выбирает подходящий раздел; осуществляет загрузку программы и настройку адресов.

При очевидном преимуществе, заключающемся в простоте реализации, данный метод имеет существенный недостаток – **жёсткость**. Так как в каждом разделе может выполняться только одна программа, то уровень мультипрограммирования заранее ограничен числом разделов независимо от того, какой размер имеют программы.

Даже если программа имеет небольшой объём, она будет занимать весь раздел, что приводит к неэффективному использованию памяти. С другой стороны, даже если объём оперативной памяти машины позволяет выполнить некоторую программу, разбиение памяти на разделы не позволяет сделать этого.

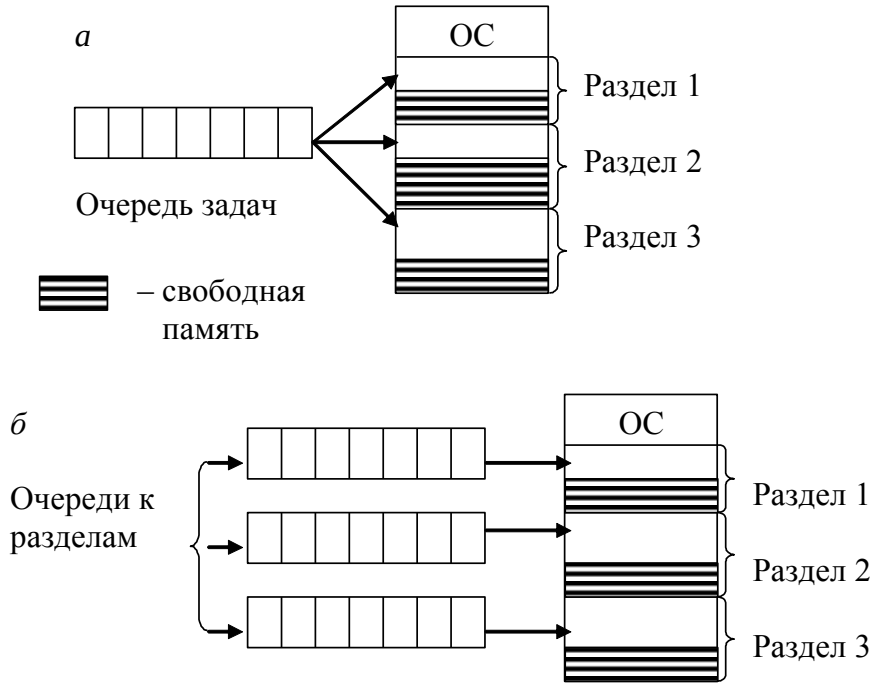


Рис. 3.15. Распределение памяти фиксированными разделами:
а – с общей очередью; *б* – с отдельными очередями

Распределение памяти разделами переменной величины

В этом случае память машины не делится заранее на разделы. Сначала вся память свободна. Каждой вновь поступающей задаче выделяется необходимая ей память. Если достаточный объём памяти отсутствует, то задача не принимается на выполнение и стоит в очереди. После завершения задачи память освобождается и на это место может быть загружена другая задача. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера. На рис. 3.16 показано состояние памяти в различные моменты времени при использовании **динамического распределения**. Так, в момент t_0 в памяти находится только ОС, а к моменту t_1 память разделена между 5 задачами, причём задача П4, завершая работу, покидает память к моменту t_2. На освободившееся место загружается задача П6, поступившая в момент t_3.

Задачами операционной системы при реализации данного метода управления памятью являются: ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти; анализ запроса (при поступлении новой задачи); просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей задачи: загрузка задачи в выделенный ей раздел и корректировка таблиц свободных и занятых областей; корректировка таблиц свободных и занятых областей (после завершения задачи).

Программный код не перемещается во время выполнения, т.е. может быть проведена единовременная настройка адресов посредством использования перемещающего загрузчика.

Выбор раздела для вновь поступившей задачи может осуществляться по разным правилам: «первый попавшийся раздел достаточного размера»; «раздел, имеющий наименьший достаточный размер»; «раздел, имеющий наибольший достаточный размер». Все эти правила имеют свои преимущества и недостатки.

По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток – **фрагментация памяти**. Фрагментация – это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объём памяти.

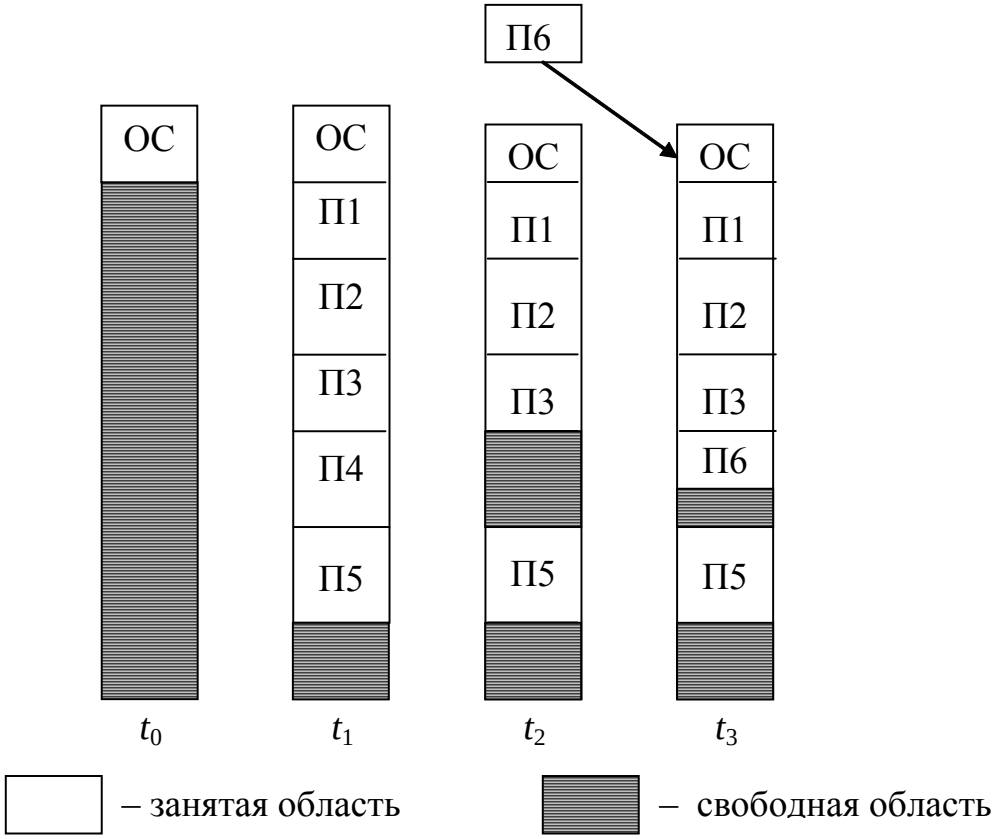


Рис. 3.16. Распределение памяти динамическими разделами

Перемещаемые разделы

Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших либо в сторону младших адресов так, чтобы вся свободная память образовывала единую свободную область (рис. 3.17). В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей.

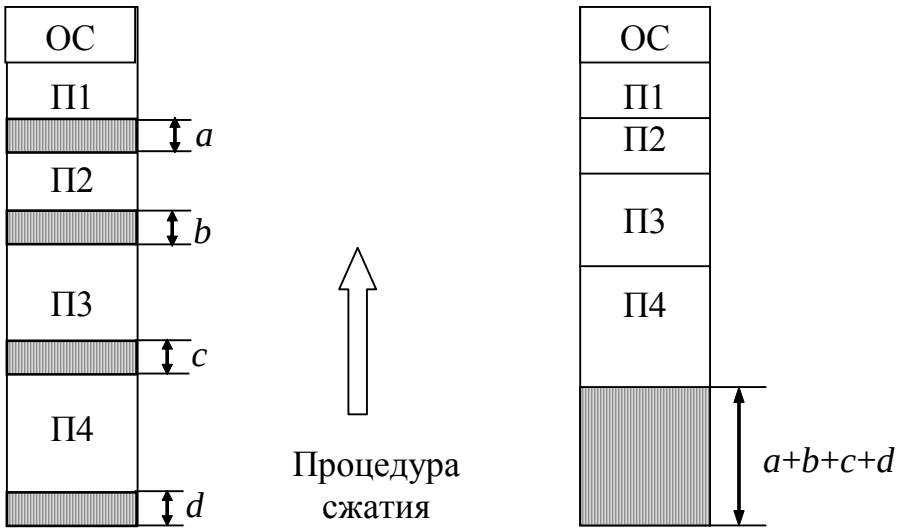


Рис. 3.17. Распределение памяти перемещаемыми разделами

Эта процедура называется **сжатием**. Сжатие может выполняться либо при каждом завершении задачи, либо только тогда, когда для вновь поступившей задачи нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц, а во втором – реже выполняется процедура сжатия. Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов из виртуальной формы в физическую должно выполняться динамическим способом.

Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества данного метода.

3.4.4 Организация виртуальной памяти

Концепция виртуальной памяти

Общепринятая в настоящее время концепция виртуальной памяти появилась достаточно давно. Она позволила решить целый ряд актуальных вопросов организации вычислений. Прежде всего к числу таких вопросов относится обеспечение надежного функционирования мультипрограммных систем. В любой момент времени компьютер выполняет множество процессов (или задач), каждый из которых располагает своим адресным пространством. Было бы слишком накладно отдавать всю физическую оперативную память какой-то одной задаче, тем более, что многие задачи реально используют только небольшую часть своего ад-

ресного пространства. Поэтому необходим механизм разделения небольшой физической памяти между различными задачами. Виртуальная память является одним из способов реализации такой возможности. Она делит физическую память на блоки и распределяет их между различными задачами, при этом она предусматривает также некоторую схему защиты, которая ограничивает задачу теми блоками, которые ей принадлежат. Большинство типов виртуальной памяти сокращают также время начального запуска программы на процессоре, поскольку не весь программный код и данные требуются ей в физической памяти, чтобы начать выполнение.

Другой вопрос, тесно связанный с реализацией концепции виртуальной памяти, касается организации вычислений на компьютере задач очень большого объёма. Раньше, если программа становилась слишком большой для физической оперативной памяти, часть её необходимо было хранить во внешней памяти (на диске) и задача приспособить её для решения на компьютере ложилась на программиста. Программисты делили программы на части и затем определяли те из них, которые можно было бы выполнять независимо, организуя оверлейные структуры, которые загружались в основную память и выгружались из неё под управлением программы пользователя. Программист должен был следить за тем, чтобы программа не обращалась вне отведённого ей пространства физической памяти. Виртуальная память освободила программистов от этого бремени.

Виртуальным называется такой ресурс, который для пользователя (пользовательской программы) представляется обладающим свойствами, которыми он в действительности не обладает. Так, например, пользователю может быть предоставлена виртуальная оперативная память, размер которой превосходит всю имеющуюся в системе реальную ОП. Пользователь пишет программы так, как будто в его распоряжении имеется однородная (одноуровневая) оперативная память большого объёма, но в действительности все данные, используемые программой, хранятся на нескольких разнородных запоминающих устройствах, обычно в ОП и на дисках, и при необходимости частями перемещаются между ними. Все эти действия выполняются автоматически, без участия программиста, т.е. механизм виртуальной памяти является прозрачным по отношению к пользователю.

Наиболее распространенными реализациями виртуальной памяти являются страничное, сегментное и странично-сегментное распределение памяти.

Страничное распределение

На рис. 3.18 показана схема страничного распределения памяти. Виртуальное адресное пространство каждого процесса делится на части, называемые **виртуальными страницами,** одинакового, фиксированного (для данной системы) размера. В общем случае размер виртуального адресного пространства не является кратным размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

Вся оперативная память машины также делится на части такого же размера, называемые **физическими страницами** (или блоками).

Размер страницы обычно выбирается равным степени двойки: 512, 1024 и т.д., это позволяет упростить механизм преобразования адресов.

При загрузке процесса часть его виртуальных страниц помещается в оперативную память, а остальные – на диск. Смежные виртуальные страницы необязательно располагаются в смежных физических страницах. При загрузке операционная система создает для каждого процесса информационную структуру – **таблицу страниц**, в которой устанавливается соответствие между номерами виртуальных и физических страниц для страниц, загруженных в оперативную память, или делается отметка о том, что виртуальная страница выгружена на диск (ВЗУ). Кроме того, в таблице страниц содержится управляющая информация, такая как признак модификации страницы, признак невыгружаемости (выгрузка некоторых страниц может быть запрещена), признак обращения к странице (используется для подсчёта числа обращений за определённый период времени) и другие данные, формируемые и используемые механизмом виртуальной памяти.

В данной ситуации может быть использовано много разных критериев выбора, наиболее популярные из них следующие:

- дольше всего не использовавшаяся страница;
- первая попавшаяся страница;
- страница, к которой в последнее время было меньше всего обращений.

В некоторых системах используется понятие рабочего множества страниц. Рабочее множество определяется для каждого процесса и представляет собой перечень наиболее часто используемых страниц, которые должны постоянно находиться в оперативной памяти и поэтому не подлежат выгрузке.

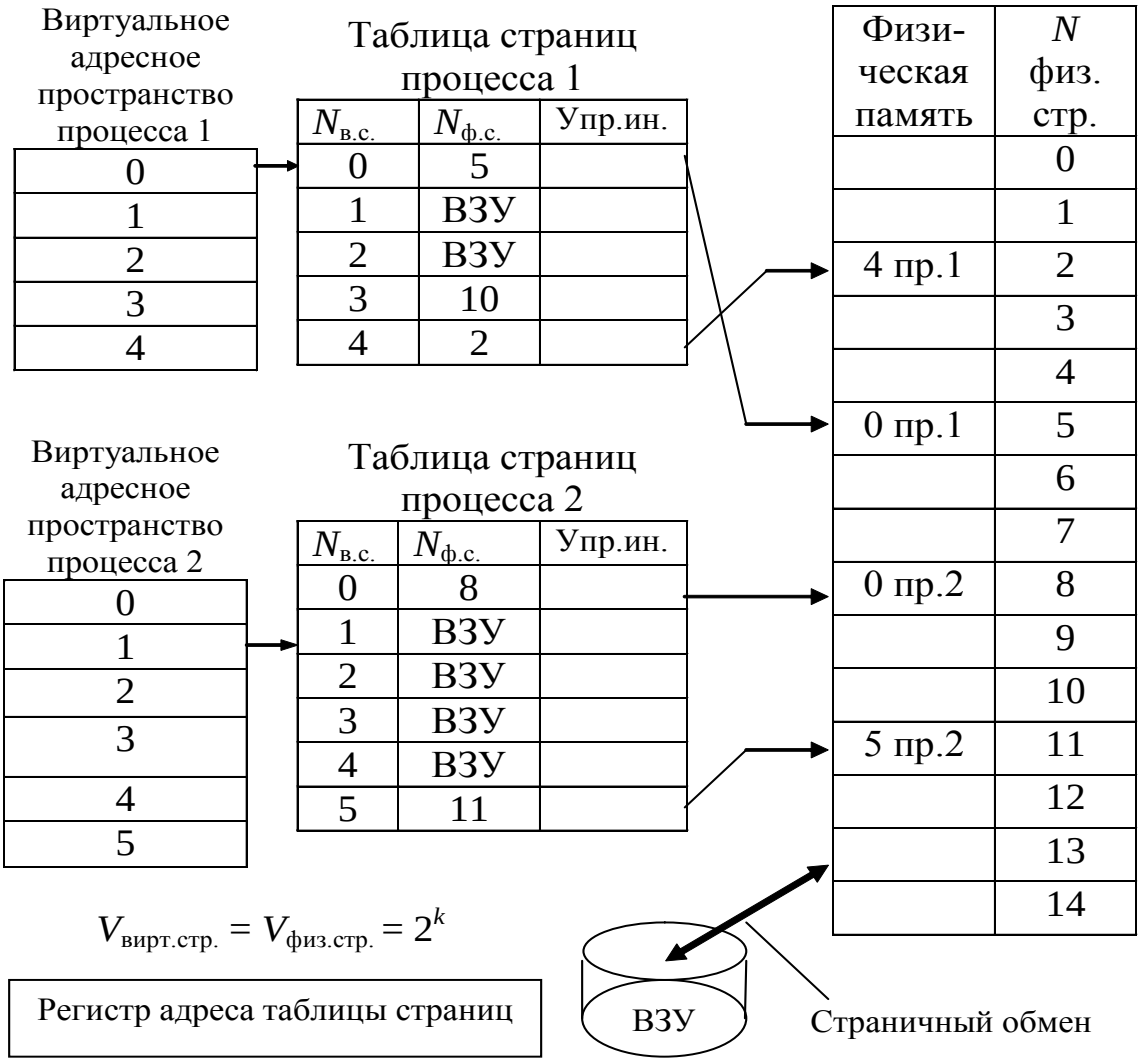


Рис. 3.18. Страничное распределение памяти

После того как выбрана страница, которая должна покинуть оперативную память, анализируется ее признак модификации (из таблицы страниц). Если выталкиваемая страница с момента загрузки была модифицирована, то ее новая версия должна быть переписана на диск. Если нет, то она может быть просто уничтожена, т.е. соответствующая физическая страница объявляется свободной.

Рассмотрим механизм преобразования виртуального адреса в физический при страничной организации памяти (рис. 3.19).

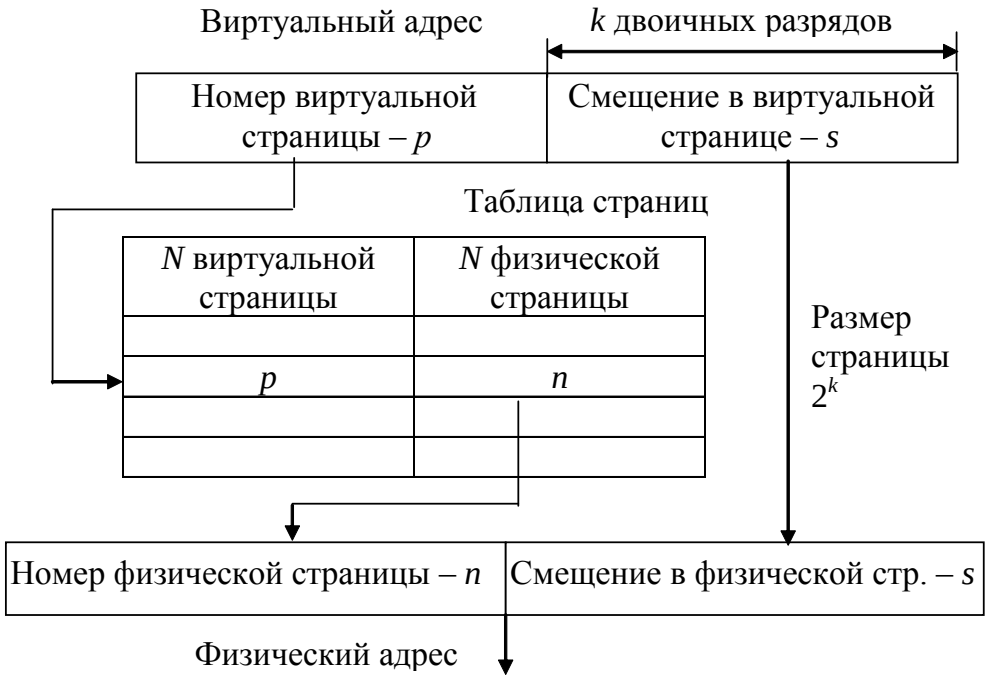


Рис. 3.19. Механизм преобразования виртуального адреса в физический при страничной организации памяти

Виртуальный адрес при страничном распределении может быть представлен в виде пары (*p, s*), где *p* – номер виртуальной страницы процесса (нумерация страниц начинается с 0), *s* – смещение в пределах виртуальной страницы. Учитывая, что размер страницы равен 2 в степени *k*, смещение *s* может быть получено простым отделением *k* младших разрядов в двоичной записи виртуального адреса. Оставшиеся старшие разряды представляют собой двоичную запись номера страницы *p*.

При каждом обращении к оперативной памяти аппаратными средствами выполняются следующие действия:

1. На основании начального адреса таблицы страниц (содержимое регистра адреса таблицы страниц), номера виртуальной страницы (старшие разряды виртуального адреса) и длины записи в таблице страниц (системная константа) определяется адрес нужной записи в таблице.

2. Из этой записи извлекается номер физической страницы.

3. К номеру физической страницы присоединяется смещение (младшие разряды виртуального адреса).

Использование в пункте (3) того факта, что размер страницы равен степени 2, позволяет применить операцию конкатенации (присоединения) вместо более длительной операции сложения, что уменьшает время

получения физического адреса, а значит, повышает производительность компьютера.

На производительность системы со страничной организацией памяти влияют временные затраты, связанные с обработкой страничных прерываний и преобразованием виртуального адреса в физический. При часто возникающих страничных прерываниях система может тратить большую часть времени впустую, на перемещение страниц. Чтобы уменьшить частоту страничных прерываний, следовало бы увеличивать размер страницы. Кроме того, увеличение размера страницы уменьшает размер таблицы страниц, а значит, уменьшает затраты памяти. С другой стороны, если страница велика, значит велика и фиктивная область в последней виртуальной странице каждой программы. В среднем на каждой программе теряется половина объема страницы, что в сумме при большой странице может составить существенную величину. Время преобразования виртуального адреса в физический в значительной степени определяется временем доступа к таблице страниц. В связи с этим таблицу страниц стремятся размещать в «быстрых» запоминающих устройствах. Это может быть, например, набор специальных регистров или память, использующая для уменьшения времени доступа ассоциативный поиск и кэширование данных.

Страничное распределение памяти может быть реализовано в упрощенном варианте, без выгрузки страниц на диск. В этом случае все виртуальные страницы всех процессов постоянно находятся в оперативной памяти. Такой вариант страничной организации хотя и не предоставляет пользователю виртуальной памяти, но почти исключает фрагментацию за счет того, что программа может загружаться в несмежные области, а также того, что при загрузке виртуальных страниц никогда не образуется остатков.

Для вычисления количества записей таблицы страниц и размера таблицы страниц можно использовать следующие выражения:

Количество записей таблицы страниц (КЗТС) = $2^n / 2^k$

где n *– разрядность виртуального адреса,*

2^k *– размер страницы в байтах*

Размер таблицы страниц = КЗТС * V

где V *–размер записи в таблице страниц*

Сегментное распределение

При страничной организации виртуальное адресное пространство процесса делится механически на равные части. Это не позволяет дифференцировать способы доступа к разным частям программы (сегментам), а это свойство часто бывает очень полезным. Например, можно запретить обращаться с операциями записи и чтения в кодовый сегмент программы, а для сегмента данных разрешить только чтение. Кроме того, разбиение программы на «осмысленные» части делает принципиально возможным разделение одного сегмента несколькими процессами. Например, если два процесса используют одну и ту же математическую подпрограмму, то в оперативную память может быть загружена только одна копия этой подпрограммы.

Рассмотрим, каким образом сегментное распределение памяти реализует эти возможности. Виртуальное адресное пространство процесса делится на **сегменты, размер которых определяется программистом** с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т.п. Иногда сегментация программы выполняется по умолчанию компилятором.

При загрузке процесса часть сегментов помещается в оперативную память (при этом для каждого из этих сегментов операционная система подыскивает подходящий участок свободной памяти), а часть сегментов размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки. Во время загрузки система создает **таблицу сегментов процесса** (аналогичную таблице страниц), в которой для каждого сегмента указывается начальный физический адрес сегмента в оперативной памяти, размер сегмента, правила доступа, признак модификации, признак обращения к данному сегменту за последний интервал времени и некоторая другая информация. Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок оперативной памяти, в который данный сегмент загружается в единственном экземпляре.

Система с сегментной организацией функционирует аналогично системе со страничной организацией: время от времени происходят прерывания, связанные с отсутствием нужных сегментов в памяти, при необходимости освобождения памяти некоторые сегменты выгружаются, при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Кроме того, при обращении

к памяти проверяется, разрешен ли доступ требуемого типа к данному сегменту.

Виртуальный адрес при сегментной организации памяти может быть представлен парой (g, s), где g – номер сегмента, а s – смещение в сегменте. Физический адрес получается путем сложения начального физического адреса сегмента, найденного в таблице сегментов по номеру g, и смещения s.

Недостатком данного метода распределения памяти является фрагментация на уровне сегментов и более медленное по сравнению со страничной организацией преобразование адреса.

Странично-сегментное распределение

Как видно из названия, данный метод представляет собой **комбинацию страничного и сегментного распределения памяти** и, вследствие этого, сочетает в себе достоинства обоих подходов. Виртуальное пространство процесса делится на сегменты, а каждый сегмент, в свою очередь, делится на виртуальные страницы, которые нумеруются в пределах сегмента. Оперативная память делится на физические страницы. Загрузка процесса выполняется операционной системой постранично, при этом часть страниц размещается в оперативной памяти, а часть на диске. Для каждого сегмента создаётся своя таблица страниц, структура которой полностью совпадает со структурой таблицы страниц, используемой при страничном распределении.

Для каждого процесса создаётся таблица сегментов, в которой указываются адреса таблиц страниц для всех сегментов данного процесса. Начальный адрес таблицы сегментов загружается в специальный регистр процессора, когда активизируется соответствующий процесс.

Виртуальный адрес при странично-сегментном распределении состоит из трёх частей (g, p, s), где g – номер сегмента, p – номер виртуальной страницы процесса, s – смещение в пределах виртуальной страницы. Трансляция виртуального адреса в физический с использованием таблиц сегментов и страниц начинается (рис. 3.17) с того, что на основании начального адреса таблицы сегментов (содержимое регистра адреса таблицы сегментов), номера сегмента (старшие разряды виртуального адреса) определяется базовый адрес соответствующей таблицы страниц для данного сегмента. А дальше происходит всё то же самое, что при страничном распределении. По найденному базовому адресу таблицы страниц, номеру виртуальной страницы p из таблицы страниц извлекается старшая часть физического адреса страницы (n), к которой присоединяется смещение s (младшая часть).

Процесс преобразования адресов посредством таблиц является достаточно длительным и, естественно, приводит к снижению производительности системы. С целью ускорения этого процесса используется специальная, полностью ассоциативная кэш-память (рис.3.17), которая называется **буфером преобразования адресов TLB (translation lookaside buffer)**.

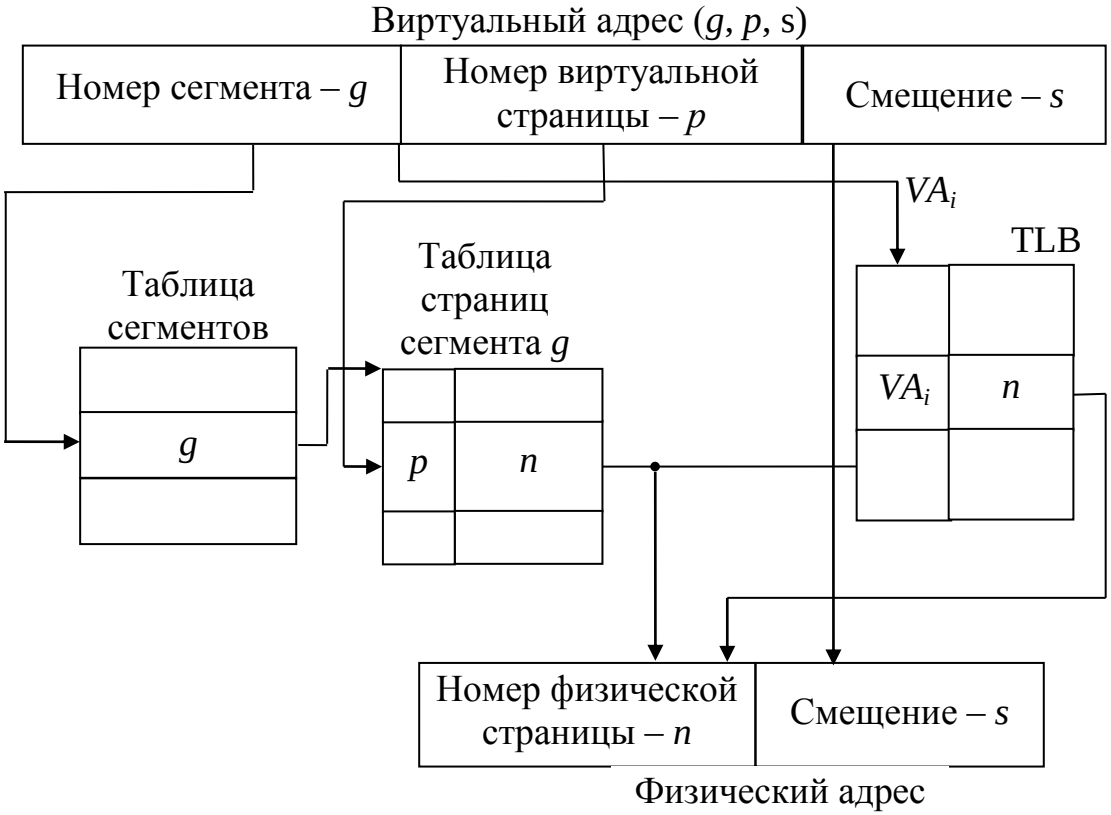


Рис. 3.20. Механизм преобразования адресов для странично-сегментной организации памяти с использованием TLB

Виртуальный адрес страницы VA_i, составленный из полей g и p, передаётся в TLB в качестве поискового признака (тега). Он сравнивается с тегами (VA) всех ячеек TLB, и при совпадении из найденной ячейки выбирается физический адрес страницы n, позволяющий сформировать полный физический адрес элемента данных, находящегося в ОП. Если совпадение не произошло, то трансляция адресов осуществляется обычными методами через таблицы сегментов и страниц. Эффективность преобразования адресов с использованием TLB зависит от коэффициента «попадания» в кэш-памяти, т.е. от того, насколько редко приходится обращаться к табличным методам трансляции адресов. Учитывая прин-

цип локальности программ и данных, можно сказать, что при первом обращении к странице, расположенной в ОП, физический адрес определяется с помощью таблиц и загружается в соответствующую ячейку TLB. Последующие обращения к странице выполняются с использованием TLB.

3.4.5 Методы ускорения процессов обмена между ОП и ВЗУ

Эффективная скорость обмена между оперативным и внешним уровнями памяти в значительной степени определяется затратами на поиск секторов или блоков в накопителе ВЗУ. Для уменьшения влияния затрат времени поиска информации на скорость обмена используют традиционные методы **буферизации и распараллеливания.** Метод буферизации заключается в использовании так называемой дисковой кэш-памяти. **Дисковый кэш** уменьшает среднее время обращения к диску. Это достигается за счёт того, что копии данных, находящихся в дисковой памяти, заносятся в полупроводниковую память. Когда необходимые данные оказываются находящимися в кэше, время обращения значительно сокращается. За счет исключения задержек, связанных с позиционированием головок, время обращения может быть уменьшено в 2–10 раз.

Дисковый кэш может быть реализован программно или аппаратно.

Программный дисковый кэш – это буферная область в ОП, предназначенная для хранения считываемой с диска информации. При поступлении запроса на считывание информации с диска вначале производится поиск запрашиваемой информации в программном кэше.

При наличии в кэше требуемой информации она передаётся в процессор. Если она отсутствует, то осуществляется поиск информации на диске. Считанный с диска информационный блок заносится в буферную область ОП (программный дисковый кэш). Программа, управляющая дисковой кэш-памятью, осуществляет также слежение и за работой диска. Весьма хорошую производительность показывают программы Smart Drv, Ncache и Super PC-Kwik. Иногда для программного кэша используется дополнительная или расширенная память компьютера.

Аппаратный дисковый кэш – это встроенный в контроллер диска кэш-буфер с ассоциативным принципом адресации информационных блоков. По запросу на считывание информации вначале производится поиск запрашиваемого блока в кэше. Если блок находится в кэше, то он передаётся в ОП. В противном случае информационный блок считывается с диска и заносится в кэш для дальнейшего использования. При поступлении запроса на запись информационный блок из ОП заносится

вначале в дисковый кэш и лишь затем (после выполнения соответствующих операций по поиску сектора) – на диск, при этом обычно копия блока в дисковом кэше сохраняется. Запись информационного блока из ОП в кэш производится на место блока, копия которого сохранена на диске. Для управления процессами копирования вводятся специальные указатели, которые определяют, сохранена ли данная копия на диске, к какому информационному блоку обращение производилось ранее других и т.п. Копирование блока на диск производится по завершении операции поиска и не связано непосредственно с моментом поступления запроса.

Второй способ, позволяющий уменьшить снижение эффективной скорости обмена, вызванное операциями поиска на диске, связан с **использованием нескольких накопителей на диске.** Все информационные блоки распределяются по нескольким накопителям, причём так, чтобы суммарная интенсивность запросов по всем накопителям была одинаковой, а запросы по возможности чередовались. Если известны интенсивности запросов к каждому информационному блоку, то можно ранжировать эти блоки, а если при этом известны и логические связи между блоками, то связанные блоки с примерно одинаковыми интенсивностями запросов должны размещаться в разных накопителях. Это позволяет совместить операции обмена между ОП и одним из накопителей с операциями поиска очередного блока в других накопителях.

3.5 Жесткие диски и твердотельные накопители

3.5.1 Накопители на жестких магнитных дисках

Устройство жестких дисков

Накопители на жестких магнитных дисках (НЖМД, Hard Disk Drive, HDD или разг. «винчестер») – накопитель информации, основанный на магнитных пластинах и эффекте магнетизма. Применяется повсеместно в персональных компьютерах, ноутбуках, планшетных компьютерах, серверах и.т.д. Содержит набор пластин (4-9), представляющих чаще всего металлические диски, покрытые магнитным материалом – **платтером** (гамма-феррит-оксид, феррит бария, окись хрома) и соединенные между собой при помощи **шпинделя** (вала, оси). Сами диски (толщина примерно 2мм.) изготавливаются из алюминия, латуни, керамики или стекла. Для записи используются обе поверхности дисков. Вал вращается с высокой постоянной скоростью (3600-7200 оборотов/мин/. Структура НЖМД приведена на рис. 3.20.

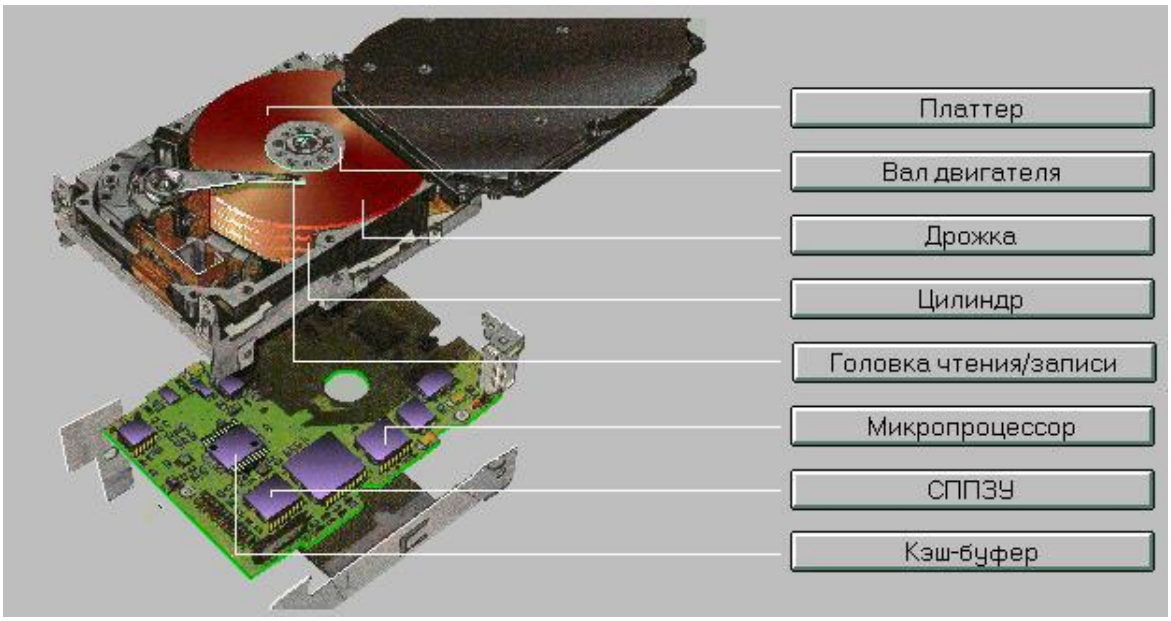


Рис. 3.21. Структура НЖМД

В современных жёстких дисках, для вала используются гидродинамические подшипники. Это даёт меньший шум при работе, значительно увеличивает долговечность и уменьшает шанс заклинивания вала из-за разрушившегося подшипника качения.

Считывание и запись производится с помощью **блока головок** (рис. 3.22).

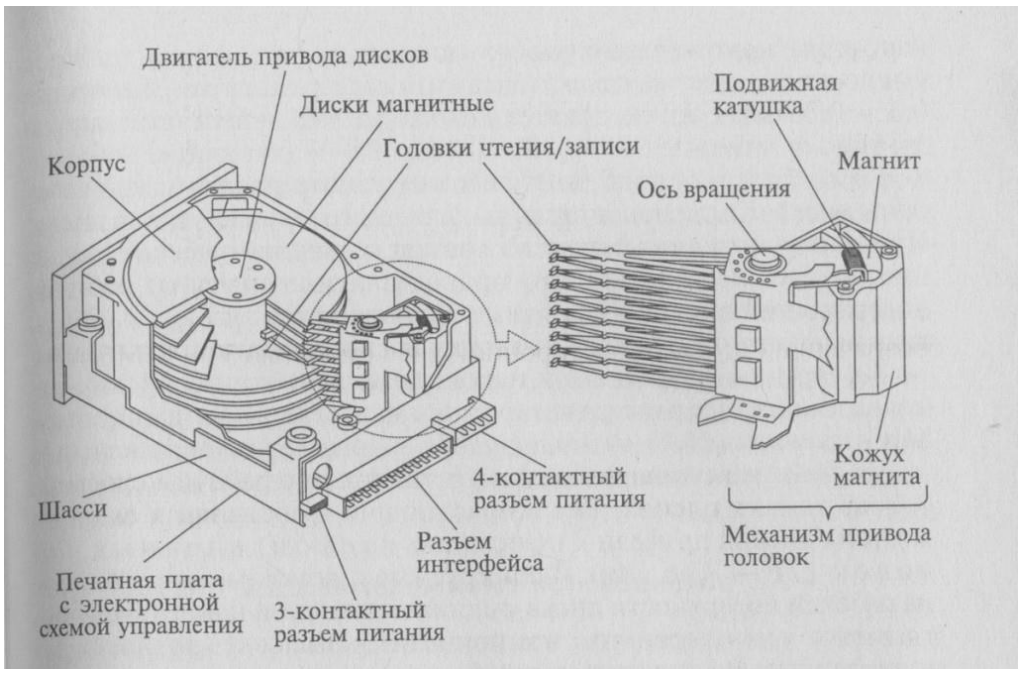


Рис. 3.22. Блок головок НЖМД

В рабочем состоянии, головки парят над поверхностью диска на расстоянии **~10нм**. Они имеют аэродинамическую форму и поднимаются над поверхностью диска за счёт восходящего потока от крутящейся пластины. Магнитные головки могут находиться с двух сторон пластины, если с каждой стороны магнитного диска нанесены магнитные слои.

Соединённый блок головок имеет **фиксированное положение**, то есть головки перемещаются все вместе. Всеми головками, управляет специальный привод, основанный на **электромагнетизме.**

Неодимовый магнит создаёт магнитное поле, в котором с высокой скоростью реакции под воздействием тока, может перемещаться блок головок. Это лучший и самый быстрый вариант перемещения блока головок, а ведь когда то блок головок перемещался механически, с помощью шестерёнок. Когда диск выключается, чтобы головки не опустились на диск и не повредили его, они убираются в зону парковки головок (**парковочная зона, parking zone,** рис. 3.23). Это также, позволяет без особых ограничений транспортировать выключенные жёсткие диски. В выключенном состоянии, диск может выдержать большие нагрузки и не повредиться. Во включенном состоянии, даже небольшой толчёк под определённым углом может разрушить магнитный слой пластины или повредить головки при касании о диск.

Рис. 3.23. Парковочная зона НЖМД

Запись информации на диск ведется по строго определенным местам — **концентрическим дорожкам** (трекам). Дорожки делятся на **сектора.** В одном секторе **512 байт** информации. Обмен данными

между ОЗУ и НМД осуществляется последовательно целым числом (**кластером**). **Кластер** — цепочки последовательных секторов (1,2,3,4).

Помимо герметичной части, у современных жёстких дисков есть наружная **плата управления**. Когда то, все платы управления были вставлены в материнскую плату компьютера в слоты расширения. Это было неудобно в плане универсальности и возможностей. Сейчас у жёстких дисков, вся управляющая диском электроника, кэш память и микроконтроллер интерфейса расположены на небольшой плате в нижней части жёсткого диска. Благодаря этому, можно настроить каждый диск под определённые, выгодные с точки зрения его строения параметры, давая ему выигрыш в скорости, либо более тихую работу к примеру.

Рис. 3.24. Плата управления НЖМД

Для подключения интерфейса и питания используются стандартные общепринятые разъёмы **PATA/SATA** и **Molex/Power SATA.**

Жёсткие диски являются самыми ёмкими хранителями информации и относительно надёжными. Объёмы дисков постоянно растут, но в последнее время это связано с некоторыми сложностями и для дальнейшего расширения объёма, требуются новые технологии. Распространению жёстких дисков в основном поспособствовало соотношение цена/объём. В большинстве случаев, гигабайт объёма диска стоит меньше чем 2.5 рубля.

Принцип записи НЖМД

В магнитных носителях информации цифровая запись производится на магнито-чувствительный материал. К таким материалам относятся некоторые разновидности оксидов железа, никель, кобальт и его соединения

При записи информации внешнее магнитное поле создается с помощью магнитной головки (рис. 3.24). В процессе считывания информации зоны остаточной намагниченности, оказавшись напротив магнитной головки, наводят в ней при считывании электродвижущую силу (ЭДС).

Изменение направления ЭДС в течение некоторого промежутка времени отождествляется с двоичной единицей, а отсутствие этого изменения — с нулем. Указанный промежуток времени называется битовым элементом.

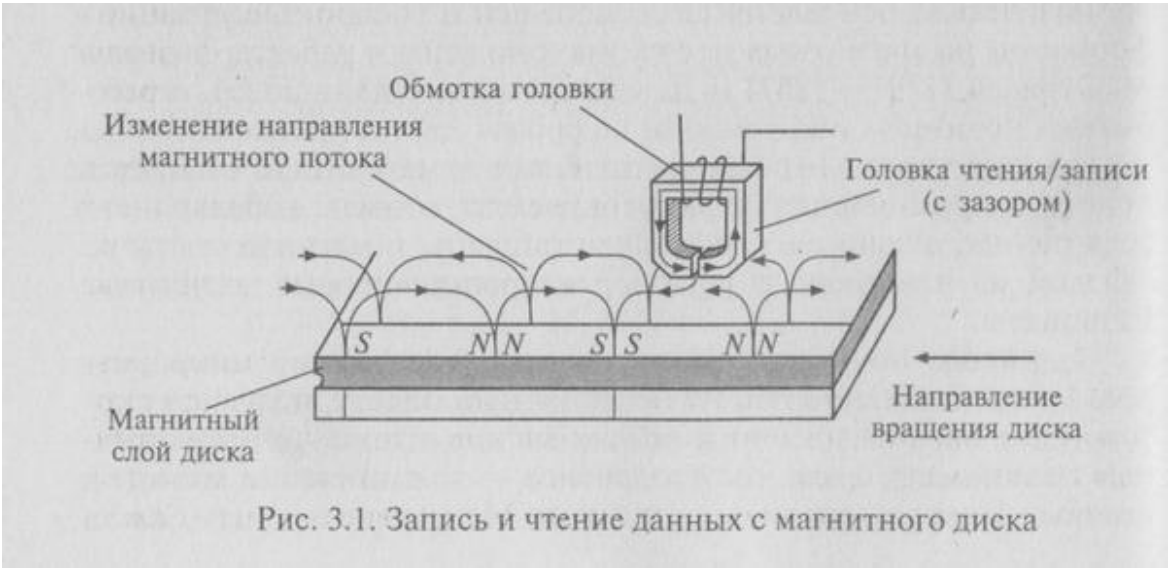


Рис. 3.1. Запись и чтение данных с магнитного диска

Рис. 3.24. Принцип записи НЖМД

Форматирование НЖМД

Поверхность магнитного носителя рассматривается как последовательность точечных позиций, каждая из которых ассоциируется с битом информации. Поскольку расположение этих позиций определяется неточно, для записи требуются заранее нанесенные метки, которые помогают находить необходимые позиции записи. Для нанесения таких синхронизирующих меток должно быть произведено разбиение диска на дорожки и секторы (рис. 3.25) — форматирование. Форматирование магнитных дисков включает 2 этапа: физическое форматирование (низкого уровня) логическое (высокого уровня).

При **физическом форматировании** рабочая поверхность диска разбивается на отдельные области, называемые секторами, которые расположены вдоль концентрических окружностей – дорожек. Также, определяются сектора, непригодные для записи данных, они помечаются как плохие для того, чтобы избежать их использования.

При **логическом форматировании** происходит окончательная подготовка носителя к хранению данных путем логической организации дискового пространства.

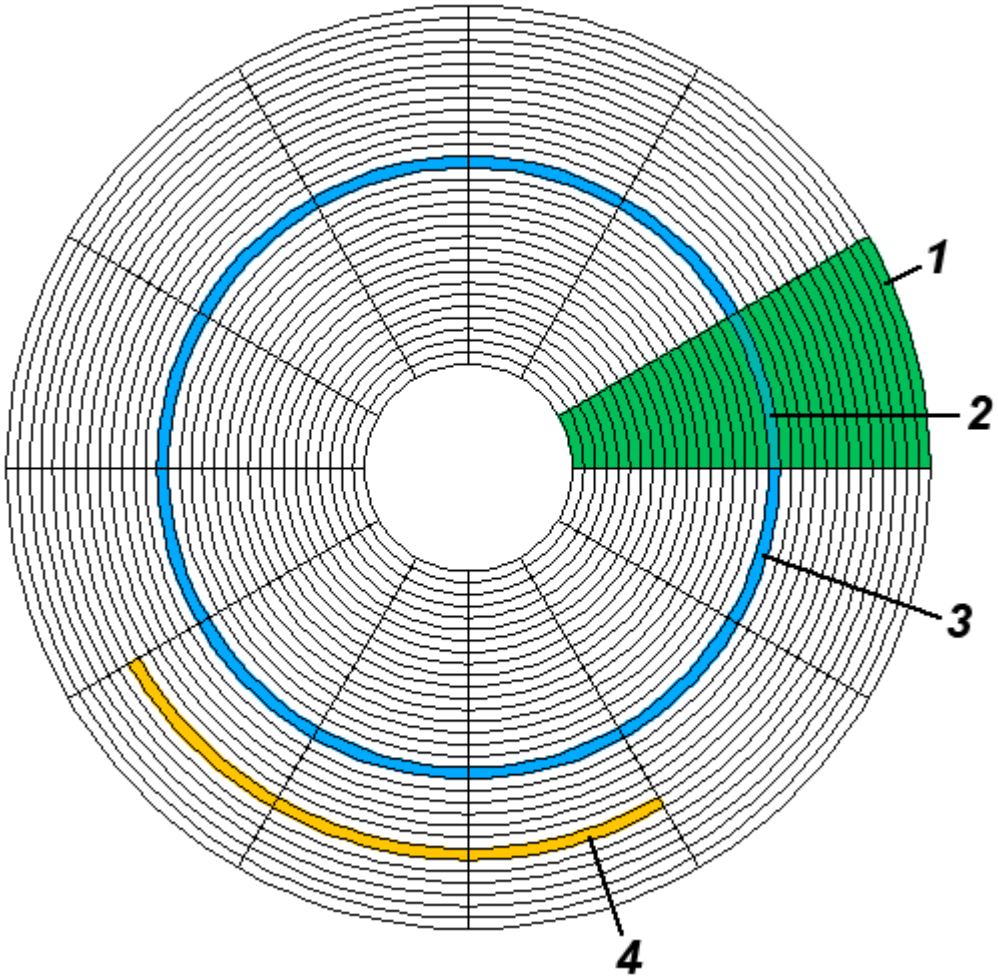


Рис. 3.25. Структура пластины НЖМД. 1 — геометрический сектор; 2 — сектор дорожки; 3 — дорожка; 4 — кластер

Характеристики НЖМД

Форм-фактор — определяет ширину жесткого диска в дюймах. Накопители имеют стандартизированные размеры 0.85, 1, 1.3, 1.8, 2.5, 3.5 дюймов. Стандартным для настольных компьютеров является 3.5 дюйма и 2.5 для ноутбуков.

Интерфейс — обеспечивает взаимодействие жесткого диска с материнской платой компьютера. В дисках предназначенных для установки внутри персональных компьютеров используется интерфейс SATA разных версий. Основное различие в скорости передачи данных: Revision 1.0 до 1,5 Гбит/с, Revision 2.0 до 3 Гбит/с (SATA/300), Revision 3.0 до 6 Гбит/с (SATA/600). Винчестеры с интерфейсом PATA (IDE) почти вышли из употребления и используются только со старым оборудованием.

Емкость — максимальное количество информации, которое может хранить жесткий диск, измеряется в гигабайтах. Поскольку производители приравнивают один килобайт к тысяче байт (на самом деле 1 Кбайт = 1 024 байт), то жесткий диск маркированный как 500 ГБ имеет реальную емкость 465.7 ГБ.

Скорость вращения шпинделя — количество оборот шпинделя в минуту. Имеет стандартные скорости, в настольных компьютерах обычно 5 400 или 7 200 об/мин, а в ноутбуках 4 500 или 5 400 об/мин. В жестких дисках для серверов скорость вращения обычно составляет 10 000 или 15 000 об/мин. Чем больше скорость вращения, тем меньше время доступа к информации.

Объем буфера — специальная высокопроизводительная память, встроенная в жесткий диск служащая для ускорения работы накопителя и сглаживания разницы в скоростях чтения/записи и передачи.

Время произвольного доступа — определяет усредненное количество времени, требуемое головке для позиционирования на произвольном участке пластины. Непостоянная величина, зависящая от начального и конечного положения головки и места позиционирования. Чем меньше данный показатель, тем быстрее диск способен отдавать запрошенную информацию.

Время наработки на отказ — среднее время безотказной работы, рассчитанное производителем. Является аналогом параметра надежность и соответственно, чем оно больше, тем лучше.

Ударостойкость — способность винчестера безболезненно переносить удары и резкую смену давления. Измеряется в единицах допустимой перегрузки, раздельно для включенного и выключенного состояния. Чем больше ударостойкость, тем лучше. Более актуально для ноутбуков и переносных винчестеров.

Уровень шума — шум, производимый жестким диском во время работы, измеряется в децибелах. Включает в себя шум вращающихся пластин, аэродинамический и шум перемещающихся головок.

3.5.2 Твердотельные накопители

Твердотельные накопители (solid-state drive, SSD) – компьютерное немеханическое запоминающее устройство на основе микросхем памяти, содержащее управляющий контроллер (рис. 3.26). Наиболее распространенный вид твердотельных накопителей использует для хранения информации флэш-память типа **NAND**, однако существуют варианты, в которых накопитель создается на **базе DRAM-памяти**, снабженной дополнительным источником питания — аккумулятором.

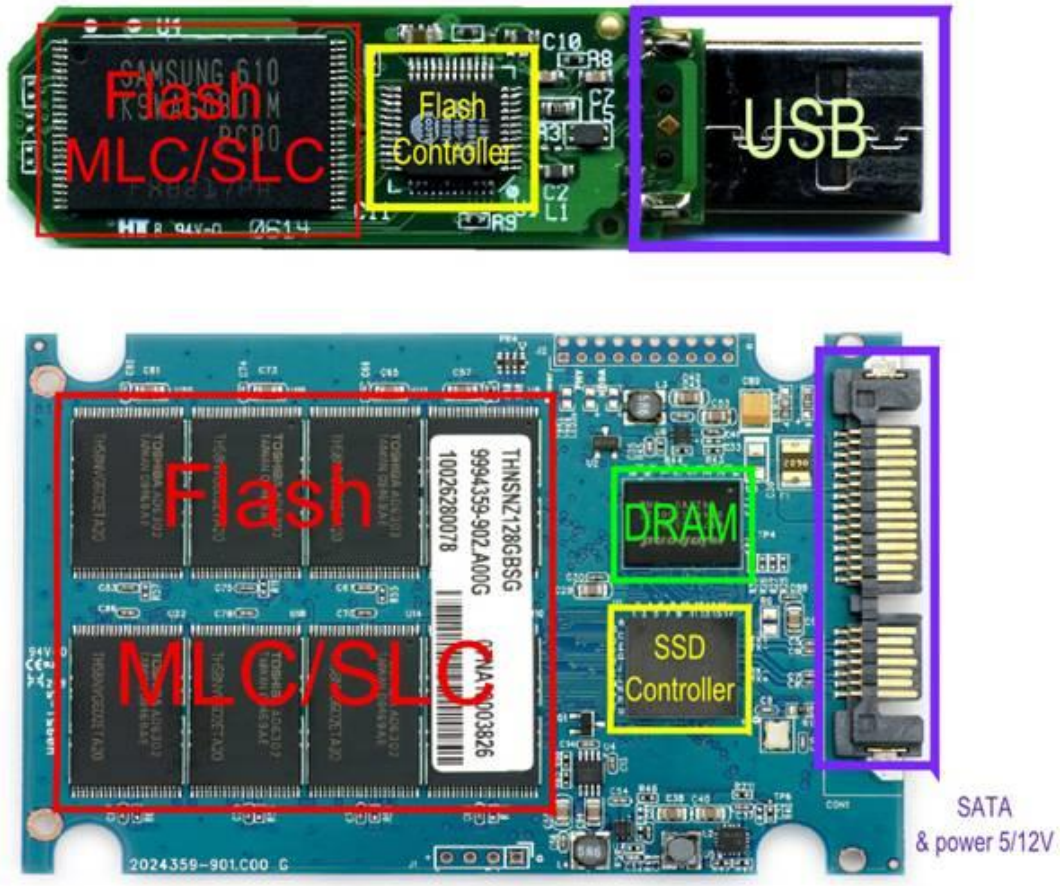


Рис. 3.26. Строение SSD

В **SSD** как и в **USB Flash** используются три типа памяти NAND: SLC (Single Level Cell), MLC (Multi Level Cell) и TLC (Three Level Cell). Отличие только в том, что SLC позволяет хранить в каждой ячейке только один бит информации, MLC – два, а TLC – три ячейки (использование разных уровней электрического заряда на плавающем затворе транзистора), что делает память MLC и TLC более дешёвой относительно ёмкости.

Микросхемы NAND флэш-памяти оптимизированы для секторного выполнения операций. Флэш-память пишется блоками по **4 Кб**, а стира-

ется по **512 Кб**. При модификации нескольких байт внутри некоторого блока контроллер выполняет следующую последовательность действий:

- считывает блок, содержащий модифицируемый блок во внутренний буфер/кэш;
- модифицирует необходимые байты;
- выполняет стирание блока в микросхеме флэш-памяти;
- вычисляет новое местоположение блока в соответствии с требованиями алгоритма перемешивания;
- записывает блок на новое место.

В **контроллер SSD** входят следующие основные элементы:

- **Processor** – как правило, 16 или 32 разрядный микроконтроллер. Выполняет инструкции микропрограммы, отвечает за перемешивание и выравнивание данных на Flash, диагностику SMART, кеширование, безопасность.
- **Error Correction (ECC)** – блок контроля и коррекции ошибок ECC.
- **Flash Controller** – включает адресацию, шину данных и контроль управления микросхемами Flash памяти.
- **DRAM Controller** - адресация, шина данных и управление DDR/DDR2/SDRAM кэш памятью.
- **I/O interface** – отвечает за интерфейс передачи данных на внешние интерфейсы SATA, USB или SAS.
- **Controller Memory** – состоит из ROM памяти и буфера. Память используется процессором для выполнения микропрограммы и как буфер для временного хранения данных. При отсутствии внешней микросхемы RAM памяти выступает в роли единственного буфера данных SSD.

Одним из ключевых параметров при измерении производительности SSD (а также НЖМД) используют термин **IOPS** (Input/Output Operations Per Second) – количество операций ввода/вывода в секунду.

IOPS – это количество блоков, которое успевает считаться или записаться на носитель. Чем больше размер блока, тем меньше «кусков», из которых состоит файл, и тем меньше будет IOPS, так как на чтение куска большего размера будет затрачиваться больше времени.

Таким образом, для определения IOPS надо знать скорость и размер блока при операции чтения / записи. Параметр IOPS равен скорости, деленной на размер блока при выполнении операции.

Основными измеряемыми величинами являются операции линейного (**последовательного**) и произвольного (**случайного**) доступа (рис. 3.27). Под **линейными** операциям чтения/записи, при которых части файлов считываются последовательно, одна за другой, подразумевается

передача больших файлов (более **128 Кб**). При произвольных операциях данные читаются случайно из разных областей носителя, обычно они ассоциируются с размером блока **4 Кбайт**.

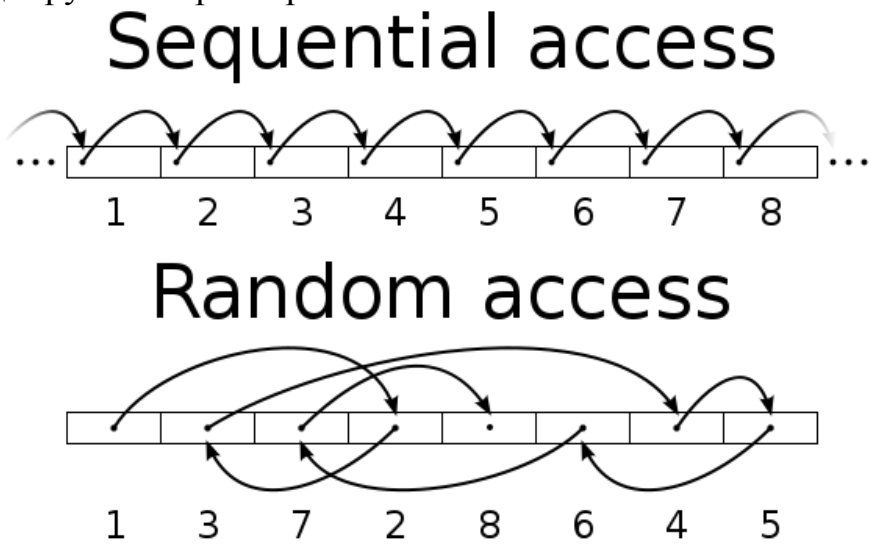


Рис. 3.27. Линейный и произвольный доступ

4 ОРГАНИЗАЦИЯ СИСТЕМНОГО ИНТЕРФЕЙСА И ВВОДА/ВЫВОДА ИНФОРМАЦИИ

4.1 Общая характеристика и классификация интерфейсов

Связь устройств ЭВМ друг с другом осуществляется с помощью интерфейсов.

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

Производительность и эффективность использования компьютера определяются не только возможностями его процессора и пропускной способностью основной памяти, но в очень большой степени характеристиками интерфейсов, составом периферийных устройств (ПУ), их техническими данными.

Объединение отдельных подсистем (устройств, модулей) ЭВМ в единую систему основывается на многоуровневом принципе с унифицированным сопряжением между всеми уровнями **– стандартными интерфейсами**. Под стандартными интерфейсами понимают такие интерфейсы, которые приняты и рекомендованы в качестве обязательных отраслевыми или государственными стандартами, различными международными комиссиями, а также крупными зарубежными фирмами.

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 (рис. 4.1) даёт возможность всем модулям ($М_1$, …, $М_n$) работать независимо с центральным модулем (ЦМ). Он позволяет получить высокие скорости передачи информации, но требует большого количества шин. **Магистральный интерфейс** (общая шина) использует принцип разделения времени для связи между ЦМ и другими модулями. Он сравнительно 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». Каждый цикл передачи сопровождается последовательностью управляющих сигналов, которые вырабатываются передающим и приёмным устройствами. Передающее устройство может осуществлять передачу данных (байта или нескольких байтов) только после подтверждения приёмником своей готовности к приёму данных.

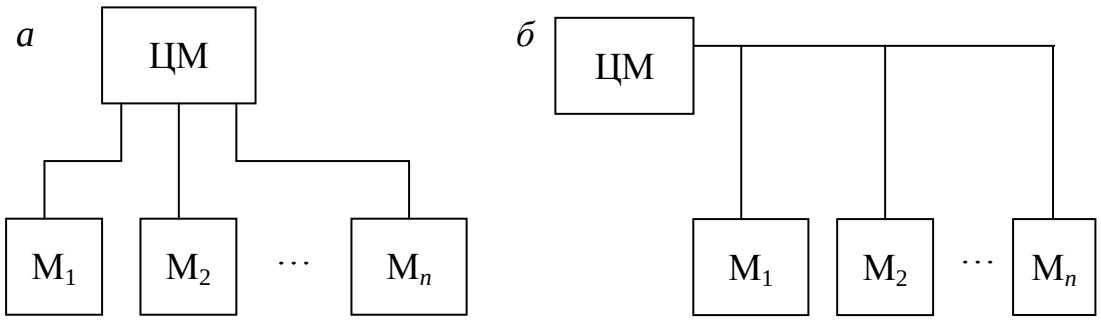


Рис. 4.1. Радиальный (*а*) и магистральный (*б*) интерфейсы

Классификация интерфейсов по назначению содержит следующие уровни сопряжений:

- **системные интерфейсы**;
- **локальные интерфейсы**;

- **интерфейсы периферийных устройств** (малые интерфейсы);
- **межмашинные интерфейсы.**

Системные интерфейсы предназначены для организации связей между центральным процессором, ОП и контроллерами (адаптерами) ПУ, а также между процессорами в многопроцессорных системах.

Локальные интерфейсы предназначены для организации связи с отдельными устройствами компьютера (видеокартой), а также 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.

Необходимость сохранения баланса производительности по мере роста быстродействия микропроцессоров привела к многоуровневой организации шин интерфейсов на основе использования **специализированных микросхем (чипсетов).**

Слово «чипсет» (chipset) в буквальном переводе означает «набор микросхем». Чипсет, который также называют **набором системной логики,** – это одна или две микросхемы, предназначенные для организации взаимодействия между процессором, памятью, интерфейсом графического адаптера, портами ввода/вывода и остальными компонентами компьютера. Со временем эти микросхемы стали называть **мостами,** появились устоявшиеся термины «северный мост» (North Bridge) и «южный мост» (South Bridge) чипсета. Если чипсет состоит из одной микросхемы, то такое решение называют одночиповым, а если из двух – двухмостовой схемой. В классической (традиционной) архитектуре двухмостового чипсета **северный мост** содержит контроллер памяти, контроллер графической шины (PCI Express), интерфейс взаимодействия с южным мостом и интерфейс взаимодействия с процессором через сокет определенного типа. Под **сокетом** понимается электрический соединитель, с помощью которого CPU компьютера соединяется с системной платой. Использование сокета позволяет при необходимости без особых проблем поменять процессор на более мощный из того же семейства. Сегодня для интеловских процессоров используются сокеты (разъемы) **в формате PGA** (pin grid array) для мобильных компьютеров и **LGA** (land grid array) – для настольных. В первом случае штыревые выводы, располагающиеся на нижней стороне корпуса процессора, устанавливаются в отверстия сокета. Во втором случае аналогично расположенные выводы процессора имеют вид плоских контактных пло-

щадок. При установке процессора в компьютер площадки CPU прижимаются к подпружиненным выводам сокета. Использование новой микроархитектуры процессоров, выпуск нового семейства CPU, повышение разрядности внешних шин и использование новых интерфейсов зачастую требуют смены сокета, а это, в свою очередь, влечёт за собой и замену чипсета.

На **южный мост** чипсета возлагается функция организации взаимодействия с устройствами ввода/вывода. Он содержит контроллеры жёстких дисков (SATA и/или PATA), сетевой контроллер, USB-контроллер, контроллер шин PCI и PCI Express, контроллер прерывания, DMA-контроллер, звуковой (аудио) контроллер. Кроме того, южный мост соединяется еще с одной важной микросхемой на материнской плате – микросхемой ROM-памяти BIOS (Basic Input-Output System – базовая система ввода/вывода). Это постоянная память, в которой хранится программа, отвечающая за базовые функции интерфейса и настройки оборудования, на котором она установлена. Наиболее широко среди пользователей компьютеров известна BIOS материнской платы, но BIOS присутствуют почти у всех компонентов компьютера: у видеоадаптеров, сетевых адаптеров, модемов, дисковых контроллеров, принтеров и т.д. Обозначение подобного базового программного обеспечения (ПО) термином «BIOS» присуще для компьютеров на базе процессоров с архитектурой x86. Для компьютеров на базе процессоров других типов для обозначения ПО, выполняющего подобные функции, используются другие термины, например, базовое ПО машин с процессором архитектуры SPARC называется PROM. Раньше к южному мосту подключалась еще одна микросхема Super I/O, которая отвечала за низкоскоростные порты RS232, LPT, RS/2. Сейчас эти функции выполняет южный мост. Для соединения северного и южного мостов друг с другом в большинстве случаев используются специальные локальные шины, причём разные производители применяют для этого разные шины с различной пропускной способностью (Intel – DMI, AMD – Alink Express, VIA – V-Link).

Чипсет является основой любой материнской платы. Фактически функциональность материнской платы и ее производительность на 90 % определяются именно чипсетом. От него зависят поддерживаемый тип процессора, тип памяти, тип сокета, а также функциональные возможности по подключению периферийных устройств. Основными компаниями на рынке чипсетов являются Intel, NVIDIA и AMD.

Шины процессора и памяти сравнительно короткие, обычно высокоскоростные и сбалансированные между собой для обеспечения мак-

симальной пропускной способности канала процессор–память. Шины ввода/вывода могут иметь большую протяжённость, поддерживать подсоединение многих типов устройств и обычно следуют одному из шинных стандартов. Обычно количество и типы устройств ввода/вывода в вычислительных системах не фиксируются (определяется количество разъёмов той или иной шины ввода/вывода), что даёт возможность пользователю самому подобрать необходимую конфигурацию. Шина ввода/вывода компьютера рассматривается как шина расширения, обеспечивающая постепенное наращивание устройств ввода/вывода. Поэтому стандарты играют огромную роль, позволяя разработчикам компьютеров и устройств ввода/вывода работать независимо.

4.2 Способы организации передачи данных

В подсистеме ввода/вывода ЭВМ используются три основных способа организации передачи данных между памятью и ПУ: программно-управляемая передача, передача по запросу прерывания от ПУ и прямой доступ к памяти (ПДП).

Программно-управляемая передача данных осуществляется при непосредственном участии и под управлением процессора, который при этом выполняет специальную подпрограмму ввода/вывода. Операция ввода/вывода инициируется центральным процессором, т.е. текущей командой программы. Данный способ является простым в реализации, но при обработке команды ввода/вывода ЦП бесполезно тратит время, ожидая готовности ПУ. Это значительно снижает производительность ЭВМ.

Второй способ передачи данных **по запросу прерывания от ПУ** реализуется под управлением контроллера прерываний (КПР) и позволяет организовывать более гибкое взаимодействие между ЦП и ПУ. Предположим, что в качестве ПУ используется клавиатура, предназначенная для ввода в ЭВМ команд, инструкций и данных. Каждый раз, когда пользователь (оператор) нажимает клавишу, ПУ выдает в КПР запрос на прерывание, который, в свою очередь, вырабатывает для ЦП сигнал прерывания. ЦП по этому сигналу приостанавливает работу текущей программы и передает управление подпрограмме ввода/вывода. Подпрограмма обрабатывает запрос и по её завершении ЦП возвращается к работе по текущей программе. Выполнение текущей программы продолжается до следующего нажатия клавиши, и далее процесс повторяется. В этом случае преимущество от использования прерывания очевидно (принципы работы системы прерывания программ описаны в разд. 2.10).

При программно-управляемой передаче данных ЦП на всё время этой передачи отвлекается от выполнения основной программы. Операция пересылки данных логически слишком проста, чтобы эффективно загружать логически сложную быстродействующую аппаратуру процессора. Вместе с тем при пересылке блока данных ЦП приходится для каждой единицы передаваемых данных (байт, слово) выполнять довольно много инструкций, чтобы обеспечить буферизацию данных, преобразование форматов, подсчёт количества переданных данных, формирование адресов в памяти и т.п. В результате скорость передачи данных при пересылке блока данных под управлением процессора оказывается недостаточной. Поэтому для быстрого ввода/вывода блоков данных и разгрузки ЦП от управления операциями ввода/вывода используют прямой доступ к памяти.

Прямой доступ к памяти

Прямой доступ к памяти (DMA – Direct Memory Access) – это такой способ обмена данными, который обеспечивает автономно от ЦП установление связи и передачу данных между ОП и ПУ. Прямой доступ к памяти освобождает процессор от управления операциями ввода/вывода, позволяет осуществлять параллельно во времени выполнение процессором программы с обменом данными между ОП и ПУ, производить этот обмен со скоростью, ограничиваемой только пропускной способностью ОП или ПУ.

Таким образом, ПДП, разгружая процессор от обслуживания ввода/вывода, способствует возрастанию общей производительности ЭВМ. Повышение предельной скорости ввода/вывода информации делает машину более приспособленной для работы в системах реального времени. Прямым доступом к памяти управляет **контроллер ПДП (DMA)** (рис. 4.2), который выполняет следующие функции:

1. Управление инициируемой процессором или ПУ передачей данных между ОП и ПУ.
2. Задание размера блока данных, который подлежит передаче, и области памяти, используемой при передаче.
3. Формирование адресов ячеек ОП, участвующих в передаче.
4. Подсчёт числа единиц данных (байт, слов), передаваемых от ПУ в ОП или обратно, и определение момента завершения заданной операции ввода/вывода.

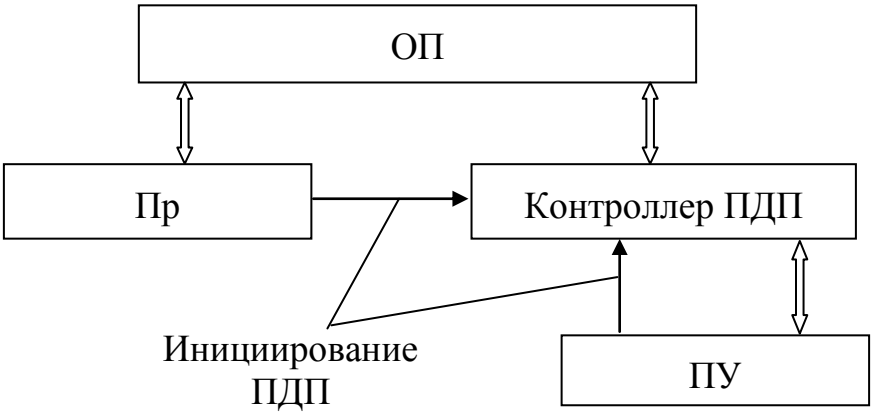


Рис. 4.2. Прямой доступ к памяти

ПДП обеспечивает высокую скорость обмена данными за счёт того, что управление обменом производится не программным путем, а аппаратурными средствами.

Контроллер ПДП обычно имеет более высокий приоритет в занятии цикла памяти по сравнению с процессором. Управление памятью переходит к контроллеру ПДП, как только завершится цикл ее работы, выполняемый для текущей команды процессора.

В современных ЭВМ используются все перечисленные способы передачи данных.

4.3 Организация интерфейса устройств ввода-вывода с процессором и операционной системой

Связь с операционной системой

Порядок передачи слова или блока по набору проводников определяется протоколом шины или сети. Но чтобы фактически произошел перенос данных из устройства в адресное пространство памяти некой пользовательской программы, остается ряд других задач, которые должны быть выполнены.

Обязанности операционной системы предопределяются тремя характеристиками систем ввода-вывода:

1. Системы ввода-вывода совместно используются множеством программ, которые выполняются процессором.

2. Системы ввода-вывода часто используют прерывания (исключения, генериру¬емые извне) для передачи информации об операциях ввода-вывода. Поскольку прерывания вызывают переход в режим ядра или супервизора, они должны обрабатываться операционной системой.

3. Низкоуровневое управление устройством ввода-вывода является довольно сложной задачей, потому что здесь требуется управление рядом параллельных событий и потому что требования по правильному управлению устройством бывают сильно детализированными.

Перечисленные 3 характеристики систем ввода-вывода приводят к тому, что операционная система должна предоставлять несколько различных функций:

- операционная система гарантирует, что пользовательская программа имеет доступ только к той части устройств ввода-вывода, на которую у пользователя есть права. Например, операционная система должна запретить программе чтение или запись файла на диске, если владелец файла не предоставил этой программе доступ к этому файлу. В системах с общими устройствами ввода-вывода защита не может обеспечиваться в том случае, если пользовательские программы могут выполнять непосредственный ввод-вывод.
- операционная система обеспечивает абстракции устройств, к которым осуществляется обращение, путем предоставления подпрограмм, управляющих низкоуровневыми операциями этих устройств.
- операционная система обрабатывает прерывания, сгенерированные устройствами ввода-вывода, точно так же, как она обрабатывает исключения, сгенерированные программой.
- операционная система старается предоставить равный доступ к общим ресурсам ввода-вывода, а также проводит диспетчеризацию доступа, чтобы улучшить пропускную способность системы.

Для выполнения этих функций по поручению пользовательских программ операционная система должна уметь связываться с устройствами ввода-вывода и не давать пользовательской программе связываться с этими устройствами напрямую. Здесь требуются три типа информационного взаимодействия:

- Операционная система должна уметь давать команды устройствам ввода-вывода. В числе этих команд должны быть не только операции вроде чтения и записи, но также и операции, которые должны быть осуществлены на устройстве, например поиск на диске.
- Устройство ввода-вывода должно уметь уведомить операционную систему о том, что оно завершило операцию или обнаружило ошибку. Например, уведомить операционную систему, когда будет завершен поиск.

- Должен быть осуществлен обмен данными между памятью и устройством ввода-вывода. Например, блок, считываемый с диска, должен быть перемещен с диска в память. Как все это делается, будет показано в нескольких следующих подразделах.

Выдача команд на устройства ввода-вывода

Для выдачи команды на устройство ввода-вывода, процессор должен уметь обратиться к устройству и предоставить ему одно или несколько командных слов. Для обращения к устройству используются два метода: **ввод-вывод с отображением на память** и **специальные инструкции ввода-вывода**. При вводе-выводе с отображением на память устройствам ввода-вывода выделяется часть адресного пространства. Чтение или запись с указанием этих адресов интерпретируется как команда устройству ввода-вывода.

Например, для отправки данных на устройство ввода-вывода может использоваться операция записи, в которой данные будут восприняты в качестве команды. Когда процессор выставляет адрес и данные на шине памяти, система памяти игнорирует операцию, поскольку адрес относится к той части адресного пространства, которая выделена для ввода-вывода. Однако контроллер устройства видит операцию, записывает данные и переносит их на устройство как команду. Пользовательские программы отстранены от непосредственного использования операций ввода-вывода, потому что операционная система не предоставляет адресное пространство, выделенное устройствам ввода-вывода, и поэтому адреса защищены преобразованием адресов. Ввод-вывод с отображением на память может также использоваться для передачи данных путем записи или чтения по избранным адресам. Устройство использует адреса для определения типа команды, и данные могут быть предоставлены с помощью записи и получены с помощью чтения. В любом случае в адресе кодируется как идентичность устройства, так и тип переноса данных между процессором и устройством.

Фактически чтение или запись данных при выполнении запроса программы обычно требует нескольких отдельных операций ввода-вывода. Кроме этого процессору между отдельными командами может потребоваться опрашивать состояние устройства, чтобы определить успешное выполнение команды. Например, у простого принтера имеются два регистра устройства ввода-вывода один для информации о состоянии, а другой для распечатываемых данных. В регистре состояния содержится бит выполнения (done bit), который устанав-

ливается принтером после распечатки символа, и бит ошибки (error bit), показывающий, что принтер зажевал бумагу или оказался без бумаги

Каждый выводимый на печать байт данных помещается в регистр данных. Затем процессор, перед тем как сможет поместить в буфер следующий символ, должен ждать, пока принтер не установит бит выполнения. Процессор должен также проверить бит ошибки, чтобы отследить возникновение проблемы. Каждая из этих операций требует отдельного обращения к устройству ввода-вывода.

Альтернативой вводу-выводу с отображением на память служит использование в процессоре специальных инструкций ввода- вывода. Эти инструкции могут определять как номер устройства, так и слово команды (или местонахождение слова команды в памяти). Процессор передает адрес устройства по набору проводников, которые обычно оставляют часть шины ввода-вывода. Текущая команда может быть передана по имеющимся на шине линиям данных. Примерами компьютеров, использующих инструкции ввода-вывода, могут служить Intel x86 и IBM 370. Пользовательские программы могут быть отстранены от непосредственного обращения к устройствам путем запрещения выполнения инструкций ввода-вывода.

Связь с процессором

Процесс периодической проверки битов состояния для определения своевременности следующей операции ввода-вывода называется опросом (polling). Опрос является простейшим способом связи устройства ввода- вывода с процессором. Устройство ввода-вывода просто помещает информацию а регистр состояния, а процессор может обратиться к нему и получить информацию. Процессор целиком вовлечен в процесс управления и делает всю работу самостоятельно.

Опрос может использоваться разными способами. Устройства ввода-вывода опрашиваются встраиваемыми приложениями реального времени, поскольку скорости ввода-вывода предопределены, что делает издержки ввода-вывода более предсказуемыми и помогает разобраться в текущем моменте.

Недостаток опроса состоит в том, что на него тратится впустую много процессорного времени, поскольку процессоры работают намного быстрее устройств ввода-вывода. Процессор может прочитать регистр состояния много раз только для того, чтобы обнаружить, что устройство еще не завершило сравнительно более медленную операцию ввода-вывода или что мышь не перемещалась со вре-

мени последнего опроса. Когда устройство завершает операцию, мы также должны считать его состояние, чтобы определить успешность этой операции.

Издержки интерфейса, построенного на опросе, стали известны очень давно, что привело к изобретению **прерываний** для оповещения процессора о том. что устройству нужно уделить внимание со стороны процессора (подробнее прерывания рассмотрены в разделе 2.10). Ввод-вывод, управляемый прерываниями, который используется почти всеми системами, у которых имеется хотя бы небольшое количество устройств, применяет прерывания ввода-вывода, чтобы показать процессору, что устройству ввода-вывода нужно уделить внимание. Когда устройство хочет уведомить процессор, что оно завершило ту или иную операцию или нуждается во внимании, оно заставляет процессор прервать его работу.

Прерывания ввода-вывода похожи на исключения, но с двумя важными отличиями:

1. По отношению к выполнению инструкций прерывание ввода-вывода является асинхронным. То есть прерывание не связано ни с одной из инструкций и не препятствует завершению инструкции. Это сильно отличается от исключения связанного с отсутствием страницы, или от исключения, вызванного арифметическим переполнением. Наш блок управления должен только проверять наличие ожидающего решения прерывания ввода-вывода к моменту запуска новой инструкции.
2. Кроме факта возникновения прерывания ввода-вывода, нужно передать еще и дополнительную информацию, например указать выдавшее его устройство. Более того, прерывания представляют устройства, которые могут иметь разные приоритеты и чьи прерывания имеют разную связанную с ними категорию срочности.

Для передачи информации процессору, например обустройстве, выдавшем прерывание, система может использовать либо векторные прерывания, либо регистр причины (Cause), используемый при исключениях. Когда процессор распознает прерывание, устройство может отправить либо адрес вектора, либо поле состояния для помещения в регистре причины. В результате, когда управление будет передано операционной системе, та будет знать об устройстве, выдавшем прерывание, и сможет немедленно детально исследовать устройство. Механизм прерываний избавляет процессор от необходимости опроса устройства, позволяя ему вместо этого сосредоточиться на выполнении программ.

4.4 Системная организация настольных компьютеров на базе современных чипсетов

4.4.1 Чипсеты Intel

В последнее время корпорации Intel удалось организовать практически полную монополию разработанных ею чипсетов для собственных процессоров. Бывшим лидерам рынка чипсетов, таким как VIA Technologies, SIS, NVIDIA, пришлось переориентироваться на разработку системной логики для других процессоров, например: AMD, VIA.

После перехода от микроархитектуры Net Burst к архитектуре Intel Core семейство чипсетов от Intel претерпело существенные изменения. Место на новых материнских платах заняла серия под кодовым именем Broadwater, которая в 2006 г. состояла из четырёх моделей: Intel Q965, Q963, G965 и P965. Эти чипсеты полностью поддерживали процессоры Core 2 Duo и работали на частоте системной шины FSB 1066 МГц.

Появившееся позже семейство чипсетов Bearlake (Intel X38, P35, G35, G33, Q35, Q33) пришло на смену предыдущего поколения микросхем и предназначалось для высокопроизводительных систем с процессорами, произведёнными по 45 нм техпроцессу.

Семейство чипсетов (Intel X58, P55, H55, H57) предназначалось для системной организации компьютеров на базе процессоров с микроархитектурой Nehalem.

С тех пор, как контроллер памяти и контроллер графической шины PCI Express переместились внутрь процессора, дизайн наборов системной логики существенно упростился. Чипсеты, состоящие ранее из пары микросхем – **северного** и **южного** мостов, переродились (начиная с Intel P55) в единый чип – **концентратор**, отвечающий за реализацию интерфейсов ввода-вывода. И теперь их обновление не оказывает существенного влияния на производительность и возможности платформы, а сказывается лишь на конструкции материнских плат, комплектуемых тем или иным набором дополнительных контроллеров.

Чипсеты шестой (P67, H67, Q67, Z68) и седьмой (Z77, Z75, H77) серий мало отличались друг от друга и использовались для процессоров Sandy Bridge и Ivy Bridge. Они поддерживали разъем LGA 1155.

Системная организация компьютеров на базе чипсетов Intel 100-й серии

Семейство чипсетов 100-й серии для настольных вариантов Skylake (шестое поколение Intel Core) включает в себя как минимум 6 модификаций: Q170, Q150, B150, Z170, H170, H110, различающихся по позици-

онированию и характеристикам. Чипсеты распределены по трем категориям. Для корпоративного сегмента используются чипсеты Q150 и Q170, которые заменили Q85 и Q87. Для сегмента малого и среднего бизнеса – чипсет B150 PCH, заменяющий южный мост B85. На потребительском уровне новый чипсет Z170 идет на смену предыдущего флагмана Z97 PCH, дополняя Q170. Вариант для массового рынка H170 заменит H97, а на начальном уровне чипсет H81 уступит место H110.

Одновременно с появлением 100-й серии чипсетов в обиход входит новый сокет – LGA 1151. По габаритам, внешнему виду и числу контактов он почти не отличается от предыдущей версии LGA 1150. Электрически, LGA 1151 имеет значительные отличия. Они связаны с появлением в Skylake поддержки памяти стандарта DDR4, удалением из процессора встроенного преобразователя питания (FIVR) и внедрением новой скоростной шины, связывающей CPU с набором системной логики.

Прошлые интеловские чипсеты, применяющиеся с процессорами поколений Haswell и Broadwell, нередко вызывают нарекания из-за того, что они и сами не предоставляют достаточное число высокоскоростных интерфейсов, и серьёзно ограничивают возможности по их добавлению в систему через внешние контроллеры. Корни этой проблемы — в шине DMI 2.0, связывающей CPU с набором логики. Её пропускная способность составляет всего 2 Гбайт/с (в каждую сторону), что ограничивает полосу пропускания, которую могут получать в своё распоряжение подключаемые через чипсет устройства.

В процессорах Skylake для настольных систем Intel наконец реализовала новую версию этой шины – DMI 3.0, которая теперь базируется на протоколе PCI Express 3.0 и имеет увеличенную до 3,9 Гбайт/с (в каждую сторону) пропускную способность. Такое изменение стало хорошим фундаментом для пересмотра функциональности наборов логики, и благодаря этому чипсеты для Skylake, имеющие кодовое имя Sunrise Point и относящиеся к сотой серии, получили заметно более широкий набор возможностей.

В качестве примера рассмотрим системную организацию процессоров на базе чипсета Intel Z170. Как видно из рисунка 4.3 этот набор логики принципиально отличается от своих предшественников появлением поддержки шины PCI Express 3.0 (8 Гбит/с) и увеличением количества портов USB 3.0. Однако, к сожалению, пока Intel воздержалась от интеграции в набор логики контроллера USB 3.1 и такие высокоскоростные порты, очевидно, будут реализовываться на платах через дополнительные чипы. Впрочем, в этом нет особой проблемы – число поддерживаемых линий PCI Express 3.0 в Intel Z170 выросло до 20, и

этого должно с лихвой хватить и на всякие добавочные контроллеры, и на разветвлённые скоростные интерфейсы для подключения твердотельных накопителей.

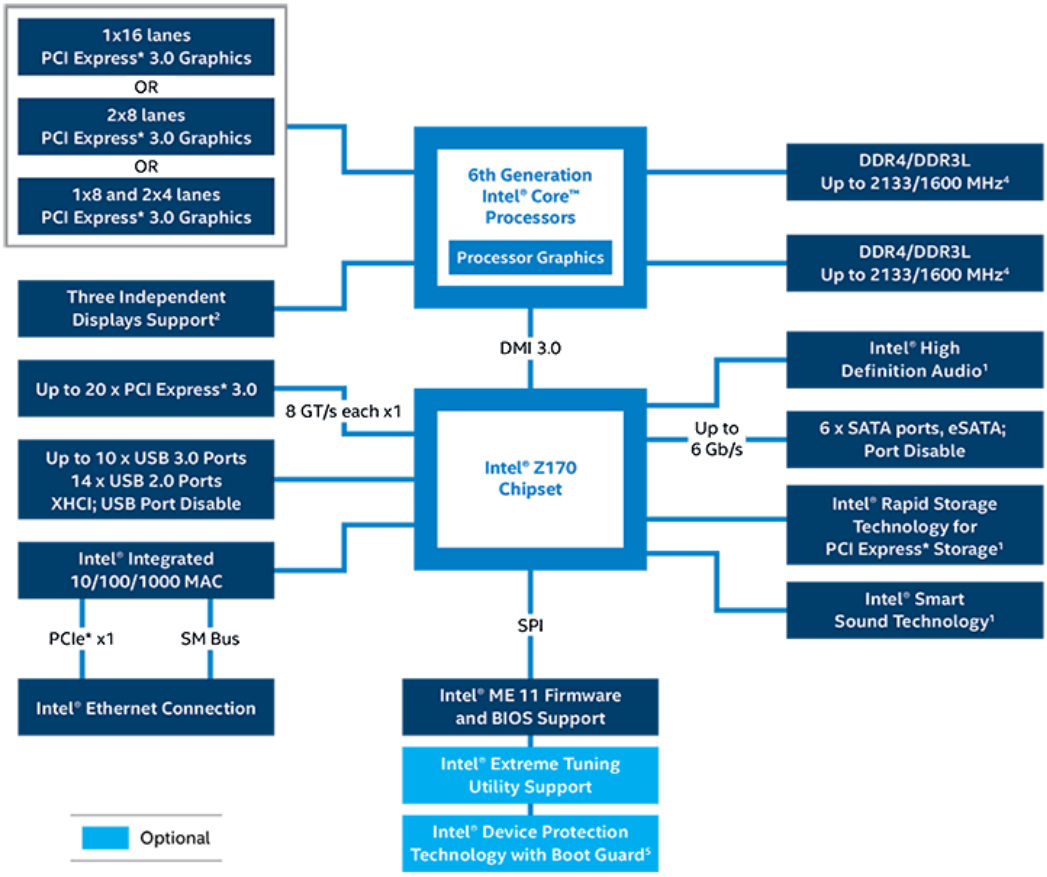


Рис. 4.3 Системная организация компьютера на базе чипсета Intel Z170

Исходя из потребностей современных интерфейсов, у типичной материнской платы на базе Z170 отведено по четыре линии PCI Express 3.0 на каждый разъём M.2 или порт U.2, по две линии – на каждую пару USB 3.1-портов и по одной линии – для каждого гигабитного LAN-контроллера. Но даже исходя из этой арифметики понятно, что 20 линий PCI Express 3.0 хватит не только на оснащение материнской платы всеми необходимыми дополнительными интерфейсами, но и на добавочные слоты PCIe 3.0 x4 или даже PCIe 3.0 x8. Иными словами, благодаря набору логики Intel Z170, производители LGA1151-плат могут придать им заметно более широкие возможности по сравнению с материнскими платами, предназначенными для Haswell или Broadwell, не прибегая к ухищрениям вроде концентраторов PCI Express.

Кстати, не стоит забывать, что помимо 20 линий PCI Express 3.0, набор логики Z170 предлагает также шесть традиционных портов SATA

(6 Гбит/с) и 10 портов USB 3.0, для реализации которых никакие дополнительные чипы вообще не требуются.

Характеристики чипсетов 100-й серии приведены в таблице 4.1.

Таблица 4.1. Характеристики чипсетов 100-й серии

	Feature/ Capability	Q170	Q150	B150	H110	H170	Z170
CHIPSET I/O	Chipset PCI Express* Gen 3 Lanes	Up to 20	10	8	6 (Gen 2 Only)	Up to 16	Up to 20
	SATA Gen 3	Up to 6	Up to 6	Up to 6	4	Up to 6	Up to 6
	USB 3.0	Up to 10	Up to 8	6	4	Up to 8	Up to 10
	Total USB Ports (USB 2.0 + 3.0)	14	14	12	10	14	14
	SATA Express Capable Ports (x2)	Up to 3	Up to 1	Up to 1	0	Up to 2	Up to 3
	Intel® RST for PCIe Storage Ports (x4 M.2 or x2 SATA Express)	Up to 3	0	0	0	Up to 2	Up to 3
	Enhanced SPI	✓	✓	✓		✓	✓
CPU	Processor PCI Express* Gen 3 1x16 Port	X4, x8, x16	1x16	1x16	1x16	1x16	X4, x8, x16

На этом преимущества новых наборов логики не заканчиваются. Важное обновление затронуло и технологию Intel Rapid Storage, возможности которой в LGA1151-чипсетах существенно расширились. Так, в ней появилась поддержка NVMe-накопителей, а, кроме того, объединять в RAID-массивы теперь можно и SSD, подключенные в систему по шине PCI Express через разъёмы SATA Express, M.2, U.2 или напрямую.

Таким образом, процессоры Skylake привлекательны не только новой микроархитектурой с возросшей удельной производительностью и поддержкой более скоростной и прогрессивной памяти, но и тем, что вся платформа в целом стала заметно лучше и функциональнее.

Системная организация компьютеров на базе чипсетов Intel 200-й серии

Одновременно с процессорами Kaby Lake-S компания Intel анонсировала и новые чипсеты Intel 200-й серии. Всего же семейство чипсетов Intel 200-й серии будет включать пять вариантов (Q270, Q250, B250, H270, Z270) для десктопных процессоров и три решения (CM238, HM175, QM175) для мобильных процессоров.

Если сопоставлять семейство новых чипсетов с семейством чипсетов 100-й серии, то Z270 — это новый вариант Z170, H270 идет на замену H170, Q270 заменяет Q170, а чипсеты Q250 и B250 заменяют Q150 и B150 соответственно. Единственный чипсет, которому не нашлось замены, это H110, т.к. в 200-й серии нет чипсета H210 или его аналога. Позиционирование чипсетов 200-й серии точно такое же, как у чипсетов 100-й серии: Q270 и Q250 ориентированы на **корпоративный рынок**, Z270 и H270 ориентированы на **пользовательские ПК**, а B250 — на **SMB-сектор** рынка.

На рис. 4.4. приведена структурная схема чипсета 200-й серии на примере Z270. Чипсеты 200-й серии позволяют комбинировать 16 процессорных портов PCIe 3.0 (PEG-портов) для реализации различных вариантов слотов PCIe. Например, чипсеты Intel Z270 и Q270 (как и их аналоги Intel Z170 и Q170) позволяют комбинировать 16 PEG-портов процессора в следующих комбинациях: x16, x8/x8 или x8/x4/x4. Остальные чипсеты (H270, B250 и Q250) допускают только одну возможную комбинацию распределения PEG-портов: x16. Также чипсеты Intel 200-й серии поддерживают двухканальный режим работы памяти DDR4 или DDR3L. Кроме того, чипсеты Intel 200-й серии поддерживают возможность одновременного подключения до трех мониторов к процессорному графическому ядру (точно так же, как и в случае чипсетов 100-й серии).

Интегрированный SATA-контроллер обеспечивает до шести портов SATA 6 Гбит/с. Поддерживается технология Intel RST (Rapid Storage Technology), которая позволяет конфигурировать SATA-контроллер в режиме RAID-контроллера (правда, не на всех чипсетах) с поддержкой уровней 0, 1, 5 и 10. Технология Intel RST поддерживается не только для SATA-портов, но и для накопителей с интерфейсом PCIe (x4/x2, разъемы M.2 и SATA Express). В топовых моделях чипсетов Intel 200-й серии поддерживается до 14 USB-портов, из которых до 10 портов могут быть USB 3.0, а остальные — USB 2.0.

Как и в чипсетах Intel 100-й серии, в чипсетах Intel 200-й серии реализована поддержка технологии Flexible I/O, которая позволяет конфигурировать высокоскоростные порты ввода/вывода (HSIO) — PCIe, SATA и USB 3.0. Технология Flexible I/O позволяет конфигурировать некоторые HSIO-порты как порты PCIe или USB 3.0, а некоторые HSIO-порты — как порты PCIe или SATA. В чипсетах Intel 200-й серии в совокупности может быть реализовано 30 высокоскоростных портов ввода/вывода (в чипсетах Intel 100-й серии было 26 HSIO-портов).

Шесть первых высокоскоростных портов (Port #1 — Port #6) строго фиксированы: это порты USB 3.0. Следующие четыре высокоскоростных

порта чипсета (Port #7 — Port #10) могут быть сконфигурированы либо как порты USB 3.0, либо как порты PCIe. Порт Port #10 при этом может использоваться и как сетевой порт GbE, то есть в сам чипсет встроен MAC-контроллер сетевого гигабитного интерфейса, а PHY-контроллер (MAC-контроллер в связке с PHY-контроллером образуют полноценный сетевой контроллер) может быть подключен только к определенным высокоскоростным портам чипсета. В частности, это могут быть порты Port #10, Port #11, Port #15, Port #18 и Port #19. Еще 12 портов HSIO (Port #11 — Port #14, Port #17, Port #18, Port #25 — Port #30) закреплены за портами PCIe. Еще четыре порта (Port #21 — Port #24) конфигурируются либо как порты PCIe, либо как порты SATA 6 Гбит/с. Порты Port #15, Port #16 и Port #19, Port #20 имеют особенность. Они могут быть сконфигурированы либо как как порты PCIe, либо как порты SATA 6 Гбит/с. Особенность заключается в том, что один порт SATA 6 Гбит/с можно сконфигурировать либо на порте Port #15, либо на порте Port #19 (то есть это один и тот же порт SATA #0, который может быть выведен либо на Port #15, либо на Port #19). Аналогично, еще один порт SATA 6 Гбит/с (SATA #1) выводится либо на Port #16, либо на Port #20.

Таким обрзом, в чипсете может быть реализовано до 10 портов USB 3.0, до 24 портов PCIe и до 6 портов SATA 6 Гбит/с. Однако, одновременно к этим 20 портам PCIe может быть подключено не более 16 PCIe-устройств. Под устройствами в данном случае понимаются контроллеры, разъемы и слоты. Для подключения одного PCIe-устройства может потребоваться один, два или четыре порта PCIe. К примеру, если речь идет о слоте PCI Express 3.0 x4, то это одно PCIe-устройство, для подключения которого требуется 4 порта PCIe 3.0.

Характеристики чипсетов 200-й серии приведены в таблице 4.2.

Таблица 4.2. Характеристики чипсетов 200-й серии

Чипсет	Q270	Q250	B250	H270	Z270
Кол-во высокоскоростных портов ввода/вывода	30	27	25	30	30
Кол-во портов PCIe 3.0	до 24	до 14	до 12	до 20	до 24
Кол-во портов SATA 6 Гбит/с	до 6	до 6	до 6	до 6	до 6
Кол-во портов USB 3.0	до 10	до 8	6	до 8	до 10
Общее кол-во USB-портов (USB 3.0 + USB 2.0)	14	14	12	14	14
Поддержка Intel RST for PCIe Storage	3	1	1	2	3
Возможные комбинации 16 процессорных портов PCIe 3.0	x16 x8/x8 x8/x4/x4	x16	x16	x16	x16 x8/x8 x8/x4/x4

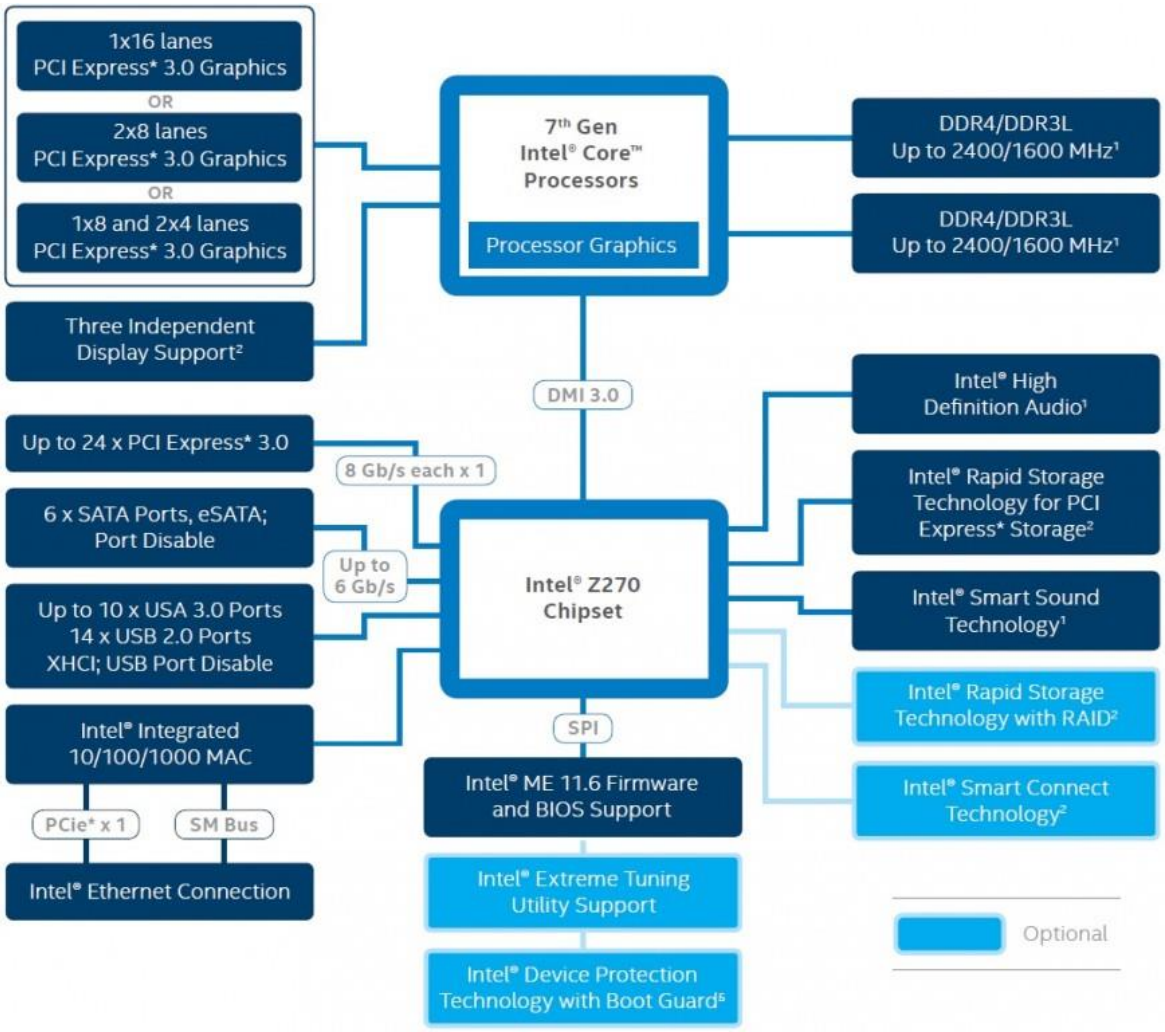


Рис. 4.4. Структурная схема чипсета 200-й серии (Z270)

Системная организация компьютеров на базе чипсетов Intel 300-й серии

В начале апреля 2018 компания Intel официально представила новые десктопные чипсеты Intel 300 series, предназначенные для процессоров Intel Core 8-го поколения.

Всего было представлено четыре набора логики: Intel H370, B360, H310 и Q370. Для обычных пользователей больший интерес представляют три первые модели. Чипсет Intel Q370 предназначен в первую очередь для корпоративных нужды и встраиваемых систем.

Среди ключевых особенностей новых наборов системной логики Intel 300 Series стоит выделить интеграцию контроллера CNVi (802.11ac, до 1,733 Гбит/с), нуждающегося во «внешнем» ИК-модуле в виде карты M.2 2230 или M.2 2216, контроллера USB 3.1 (отсутствует у

H310) и поддержку новой технологии улучшения качества звука Intel Smart Sound.

В таблице 4.3 приведены характеристики и отличия чипсетов 300-й серии.

Таблица 4.3. Характеристики чипсетов 200-й серии

Чипсеты Intel 300-й серии					
	Intel Z370	Intel H370	Intel B360	Intel H310	Intel Q370
Дата релиза	4 квартал '17	2 квартал '18			
Частота шины / версия шины	8 GT/s DMI3			5 GT/s DMI2	8 GT/s DMI3
Поддержка оверклокинга	да	Нет			
Линии PCIe	24, PCIe 3.0	20, PCIe 3.0	12, PCIe 3.0	6, PCIe 2.0	24, PCIe 3.0
Конфигурации PCIe (линии чипсета)	x1, x2, x4				
Конфигурации PCIe (линии CPU)	1x16 или 2x8 или 1x8+2x4	1x16	1x16	1x16	1x16 или 2x8 или 1x8+2x4
Поддержка оперативной памяти	2 канала	2 канала	2 канала	1 канал	2 канала
USB 2.0	до 14		до 12	до 10	до 14
USB 3.0	до 10	нет	нет	нет	нет
USB 3.1 Gen 1	нет	до 8	до 6	до 4	до 10
USB 3.1 Gen 2	нет	до 4	до 4	нет	до 6
Порты SATA	6			4	6
Поддержка локальной сети	да				
Беспроводная сеть (Intel CNVi)	нет	да			
Поддержка дисплеев	до 3			до 2	до 3
Поддержка Intel Optane	да			нет	да
Теплопакет	6 Вт				

Базовым чипсетом для процессоров Coffee Lake становится Intel H310. Оценивая спецификацию PCH, можно говорить, что это минимально улучшенная версия долгожителя Intel H110. Общее количество линий I/O увеличено с 10 до 14, плюс добавлена поддержка Integrated Intel Wireless-AC. В остальном функциональность сохранена. Это все те же 6 линий PCI Express 2.0, поддержка 10 портов USB, четыре из которых могут соответствовать USB 3.1 Gen1 (USB 3.0) и 4 канала SATA 6 Гб/с. Как и предшественник, Intel H310 позволит задействовать двухканальный режим доступа к оперативной памяти, при этом используя по одному модулю на канал. На такую базовую функциональность можно рассчитывать, выбирая самые доступные материнские платы для процессоров Intel Core 8-го поколения.

Intel B360 для чипов Coffee Lake является правопреемником очень удачного PCH, используемого для процессоров предыдущих поколений Kaby Lake/Skylake – Intel B250. Количество линий PCI Express 3.0 и каналов SATA осталось на прежнем уровне – 12 и 6, соответственно. Общее количество портов USB не изменилось – 12, но 6 из них могут соответствовать стандарту USB 3.1. При этом четыре из шести портов могут быть USB 3.1 Gen2 с пропускной способностью до 10 Гб/с. Один из

скоростных накопителей M.2 может обслуживаться Intel RST, а платы на Intel B360 позволяют использовать Intel Optane Memory, а сам чипсет потенциально готов к практической реализации Intel Wireless-AC и Intel Smart Sound.

Intel H370 предлагает еще больше линий PCI Express 3.0 и скоростных портов USB. Для обновления корпоративных платформ производитель также представил чипсет Intel Q370. Структурная схема чипсета 300-й серии на примере Z370 приведена на рис. 4.5.

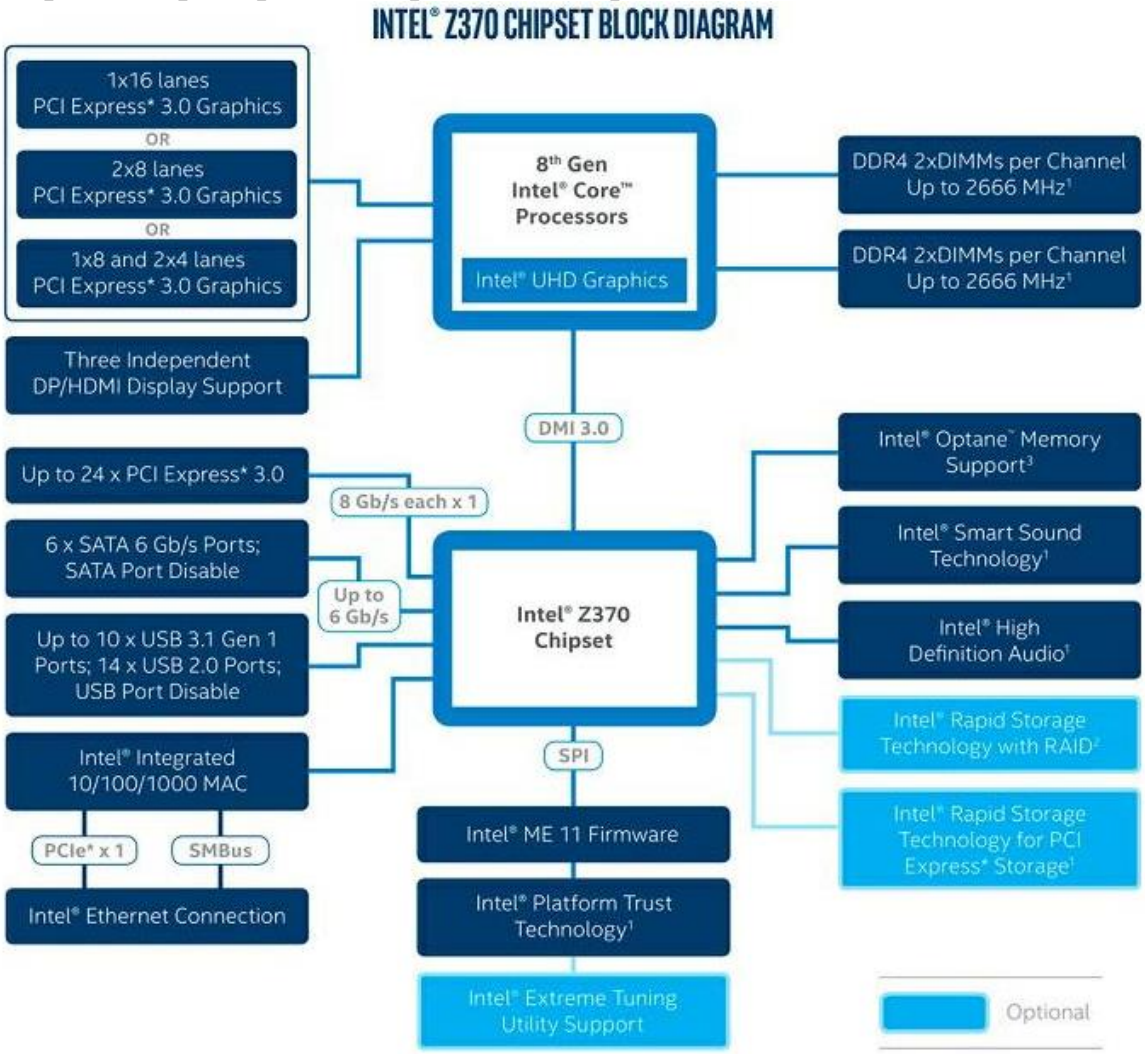


Рис. 4.5. Структурная схема чипсета 300-й серии (Z270)

4.4.2 Чипсеты AMD

Как известно, основным конкурентом в производстве процессоров общего назначения является компания AMD. Соответственно для своих

процессоров AMD выпускают собственные микросхемы и наборы логики.

На данный момент компания AMD выпускает чипсеты в двух различных конфигурациях — отдельно северный и южный мост; один объединенный чипсет. Объединенные чипсеты выпускаются под процессоры на новых сокетах AM4 и TR4, под остальные же устаревающие процессорные сокеты пока сохранилась раздельная конфигурация (северный и южный мост) чипсета.

Чипсеты AMD 9-ой серии

Чипсеты под процессоры микроархитектуры Bulldozer, Steamroller, Excavator на сокете AM3+ и его вариациях выполняются на двух отдельных микросхемах — южный мост + северный мост. Всего для настольных ПК представлено 4 чипсета: AMD 990FX, AMD 990X, AMD 980G, AMD 970.

Из чипсетов под игровые платформы представлены две вариации — AMD 990FX и 990X. Обе из них поддерживают технологию разгона и аппаратного контроля OverDrive, первый чипсет 990FX поддерживает четыре, а второй chipset 990X — две видеокарты в режиме AMD CrossFireX™ technology. Оба данных чипа не имеют встроенной графики. Также имеется похожий чип с поддержкой того же ПО OverDrive — AMD 970, к которому можно подключить только одну графическую карту.

Отдельно необходимо выделить чипсет AMD 980G с интегрированной графикой ATI Radeon™ HD 4250, который является довольно распространенным вариантом для офисных/рабочих компьютеров без встроенной видеокарты.

В таблице 4.4 приведены характеристики чипсетов AMD 9-ой серии. На рис. 4.6 приведена структурная схема чипсета AMD 9-ой серии.

Таблица 4.4. Характеристики чипсетов AMD 9-ой серии

Чипсет	Видеоадаптер	Совместимость с ЦП	Поддержка памяти	PCI Express	USB	SATA
990FX	Не входит в комплект поставки До 4-х разъемов для обновления видеокарты	AMD Athlon™ II, AMD Phenom™ II, AMD FX	DDR3	PCI Express® 2.0	До 14 USB 2.0	SATA 6 Гбит/с
990X	Не входит в комплект поставки До 2-х разъемов для обновления видеокарты	AMD Athlon™ II, AMD Phenom™ II, AMD FX	DDR3	PCI Express® 2.0	До 14 USB 2.0	SATA 6 Гбит/с
980G	ATI Radeon™ HD 4250 Поддержка DirectX 10.1 1 разъем для обновления видеокарты	AMD Athlon™ II, AMD Phenom™ II, AMD FX	DDR3	PCI Express® 2.0	До 14 USB 2.0	
970	Не входит в комплект поставки 1 разъем для обновления видеокарты	AMD Athlon™ II, AMD Phenom™ II, AMD FX	DDR3	PCI Express® 2.0	До 14 USB 2.0	SATA 6 Гбит/с

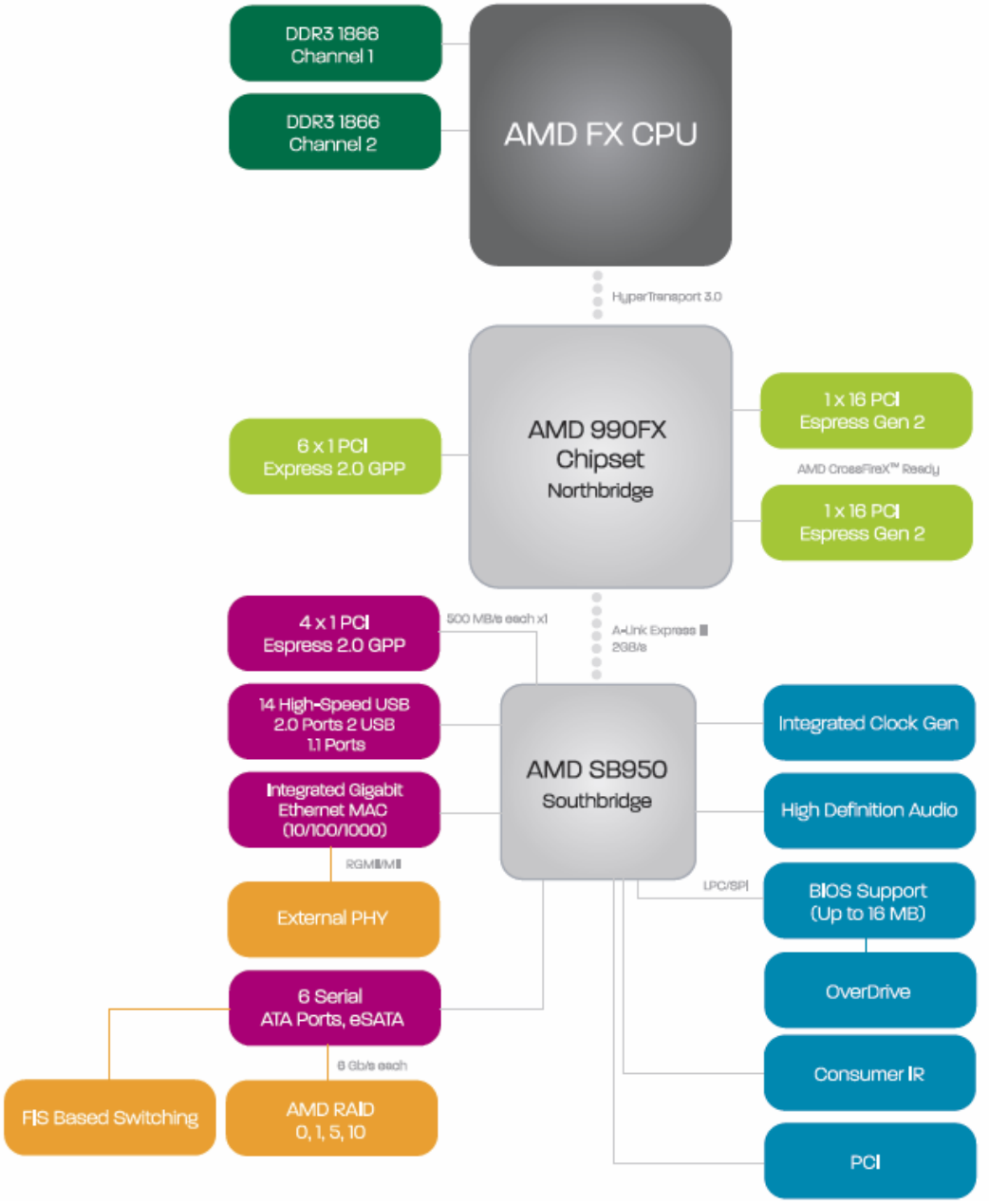


Рис. 4.6. Структурная схема чипсета AMD 9-ой серии

Чипсеты AMD 300-ой серии

Чипсеты под процессоры микроархитектуры Zen на сокете AM4 (заявленная поддержка данного сокета как минимум до 2020 года) выпускаются в варианте с одним чипом (без северного моста). Чипсет подключается к процессору по 4 линиям PCI-Express 3.0 (пропускная способность ~4ГБ в секунду). На рис. 4.7 приведена структурная схема чипсета AMD 300-й серии. В таблице 4.5 приведены характеристики чипсетов AMD 300-й серии.

Таблица 4.5. Характеристики чипсетов AMD 300-й серии

	PCI-линии	USB 2.0	USB 3.1 Gen 1	USB 3.1 Gen 2	Конфигурации PCI	Поддержка оперативной памяти	Разгон ЦП	Несколько ГП	Поддержка NVMe	RAID	Разъёмы SATA 3	Разъёмы SATA Express
X370	8 линий PCIe 2.0	6	6	2	1 x 16 или 2 x 8	Двухканальная DDR4-2666	Есть	Есть	x2	0/1/10	6	2
X300	4 линий PCIe 3.0	Нет	4	Нет	1 x 16					0/1	2	1
B350	6 линии PCIe 2.0	6	2	2				Нет		0/1/10	4	2
A320	6 линий PCIe 2.0	6	2	1			Нет			0/1/10	4	2
A300	4 линии PCIe 3.0	Нет	4	Нет						0/1	2	1

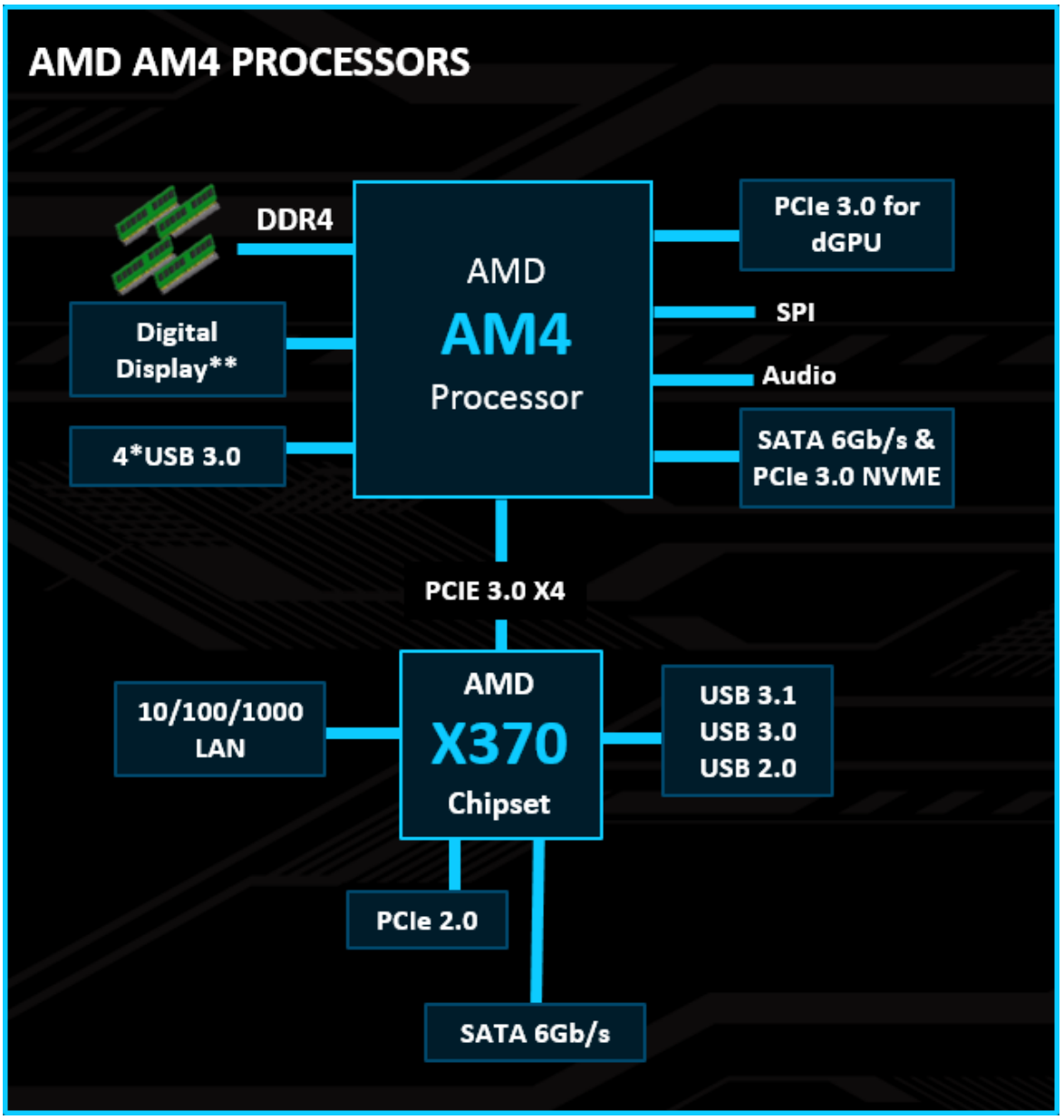


Рис. 4.7. Структурная схема чипсета AMD 300-ой серии

4.5 RAID-массивы в системе ввода-вывода

Увеличение производительности ввода-вывода стало исходной мотивацией создания дисковых массивов. В конце 1980-х годов из высокопроизводительных устройств хранения информации можно было выбрать большие, дорогостоящие диски. Аргумент состоял в том, что, заменив несколько больших дисков множеством небольших, можно улучшить производительность, поскольку в этом случае будет больше считывающих головок. Это изменение вполне соответствует множеству процессоров, поскольку наличие множества головок чтения-записи означает, что система хранения может поддерживать намного больше независимых обращений, а также длинные передачи, данные для которых разбросаны по нескольким дискам. Таким образом, можно получить как большее количество операций ввода-вывода в секунду, так и более высокую скорость передачи данных. Вдобавок к высокой производительности может быть получено преимущество в стоимости, потребляемой мощности и пространстве размещения, поскольку диски меньших размеров имеют, как правило, более высокую эффективность на гигабайт, чем более крупные диски.

В качестве отрицательного аргумента утверждалось, что дисковые массивы могут снизить надежность. Эти небольшие, недорогие накопители имеют более низкие уровни MTTF (Mean time to failure, средняя наработка на отказ), чем большие накопители, но, что более важно, за счет замены одного накопителя, скажем, пятьюдесятью небольшими накопителями интенсивность отказов возрастает как минимум в 50 раз!

Решением стало добавление избыточности, чтобы система могла справляться с отказами дисков без потери информации. При наличии множества небольших дисков расходы на дополнительную избыточность для повышения готовности невелики относительно расходов, которые требуются для решений, связанных с применением небольшого количества крупных дисков. Таким образом, безотказность была более доступной по цене при создании избыточного массива недорогих дисков. Это наблюдение легло в основу его названия: избыточные массивы недорогих дисков — **Redundant Arrays of Inexpensive Disks**, сокращенно **RAID**.

В ретроспективе, хотя их изобретение было мотивировано производительностью, главной причиной широкой популярности RAID-массивов стала безотказность. Параллельная революция вновь выдвинула на первый план исходный аргумент производительности RAID-массивов. На рис. 4.8 показано развитие RAID-массивов и их примерная

стоимость, выраженная в количестве дополнительных контрольных дисков. Чтобы отслеживать развитие RAID-массивов, авторы пронумеровали стадии этого развития, и эти номера используются по сей день.

Рис. 4.8. RAID-массивы для примера с 4-мя дисками данных, в котором показываются дополнительные контрольные диски для каждого уровня RAID

Без избыточности (RAID 0)

Простое распределение данных по нескольким дискам, называемое чередованием (striping), автоматически вынуждающее обращаться к нескольким дискам (рис. 4.9). Разделение но набору дисков заставляет всю коллекцию появляться для программного обеспечения в виде одного большого диска, что упрощает управление хранением данных. Оно также повышает производительность больших обращений, поскольку одновременно может работать сразу несколько дисков. К примеру, системы редактирования видео часто чередуют свои данные и не слишком заботятся об отказоустойчивости по сравнению с базами данных.

Название **RAID 0** употребляется неправомерно, поскольку в этом массиве отсутствует избыточность. Но уровни RAID-массивов часто

указываются оператору для настройки, и RAID 0 часто присутствует в качестве одного из вариантов. Этим и объясняется широкое распространение названия RAID 0.

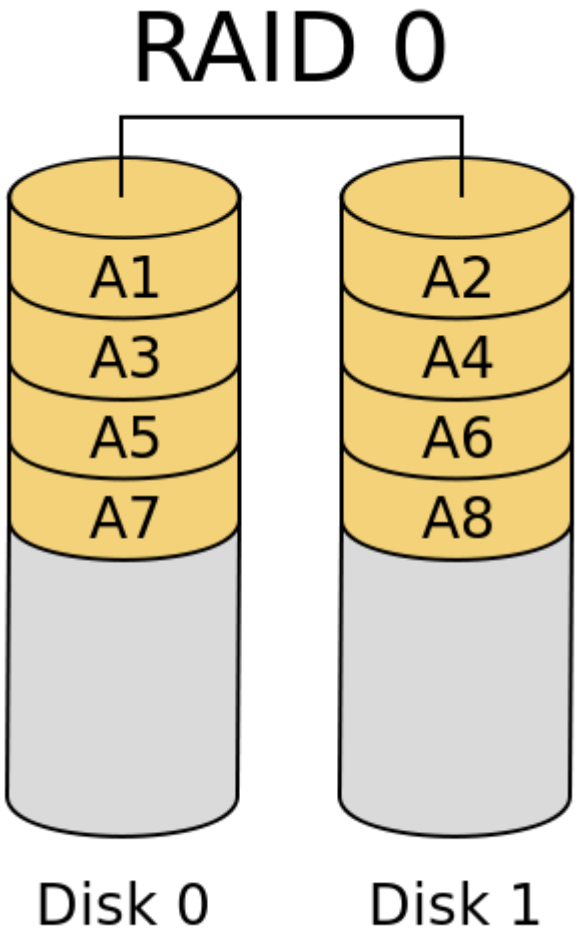


Рис. 4.9. RAID 0

Зеркальное копирование (RAID 1)

Эта традиционная схема, допускающая отказ диска, называется зеркалированием или экранированием и использует в два раза больше дисков, чем RAID 0. Как только данные записываются на один диск, эти же данные также записываются на избыточный диск, чтобы всегда было две копии информации (рис. 4.10). Если диск отказывает, система просто переходит на «зеркало» и считывает его содержимое для получения желаемой информации Зеркалирование является наиболее затратным RAID-решением, поскольку для него требуется наибольшее количество дисков.

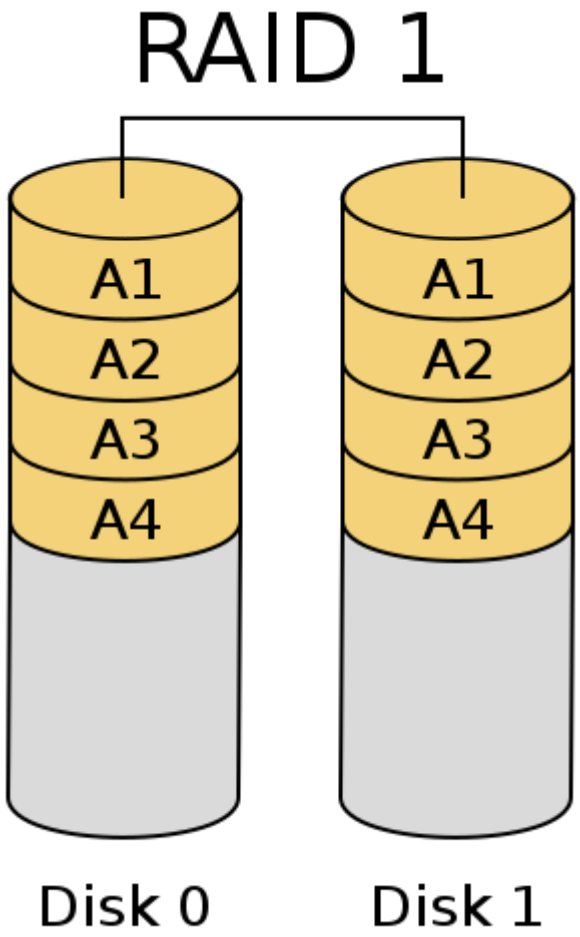


Рис. 4.10. RAID 1

Обнаружение ошибок и корректирующий код (RAID 2)

Массив RAID 2, перенимающий схему обнаружения и исправления ошибок, наиболее часто используемую для устройств памяти. В отличие от остальных уровней RAID 2 сегодня не пользуется таким широким распространением по той причине, что он представляет собой довольно громоздкую конструкцию. Архитектура данного массива основана на **коде Хемминга**, когда одни винчестеры используются для хранения информации в то время, как другие используются для хранения кодов коррекции ошибок. Таким образом, к примеру, при наличии 60 дисков в таком массиве перерасход будет составлять 19%.

Диски делятся на две группы: для данных и для кодов коррекции ошибок (рис. 4.11). Данные распределяются по дискам, предназначенным для хранения информации, так же, как и в RAID 0, то есть они разбиваются на небольшие блоки по числу дисков. Оставшиеся диски хра-

нят коды коррекции ошибок, по которым в случае выхода какого-либо жёсткого диска из строя возможно восстановление информации..

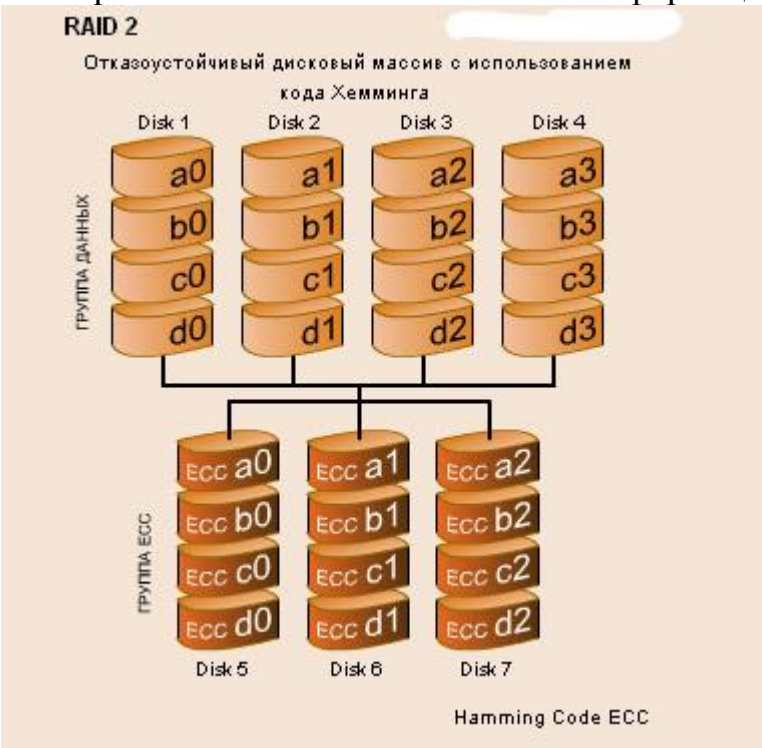


Рис. 4.11. RAID 2

Чётность с чередованием битов (RAID 3)

Стоимость более высокой готовности может быть снижена на 1/n, где n — это количество дисков в группе защиты Вместо хранения полной копии исходных данных для каждого диска нужно добавить лишь строго необходимое количество избыточной информации для восстановления утраченных при отказе данных. В операциях чтения или записи участвуют все диски группы и еще один диск, хранящий контрольную информацию на случай отказа. Массив RAID 3 популярен в приложениях с большими наборами данных, например в мультимедийных и некоторых научных программах.

Избыточный диск представляет собой хранилище суммы всех данных, имеющихся на других дисках (рис. 4.12). При отказе диска все данные из исправных дисков вычитаются из данных диска четности; оставшаяся информация должна быть идентична утраченной. Четность — это просто сумма по модулю два.

В отличие от RAID 1, для определения утраченных данных должны быть прочитаны несколько дисков. В основе этой технологии лежит предположение о приемлемости компромисса между более продолжи-

тельным временем восстановления после отказа и меньшими затратами на хранение избыточных данных.

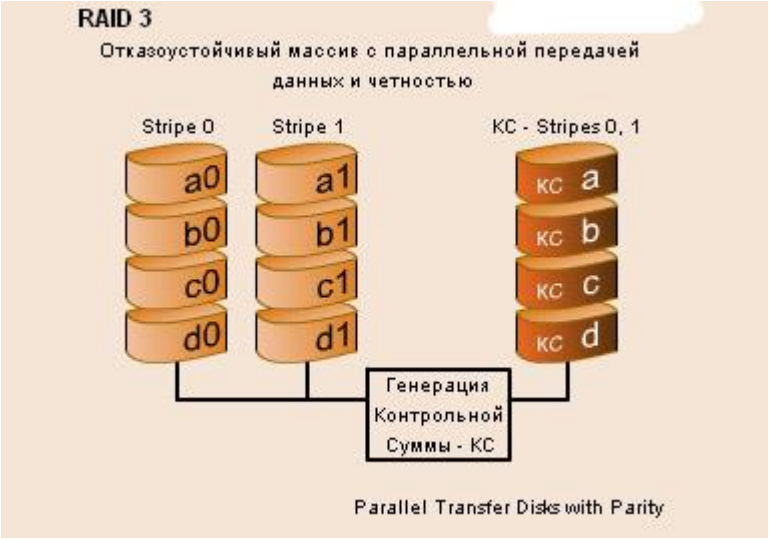


Рис. 4.12. RAID 3

Четность с чередованием блоков (RAID 4)

Массив RAID 4 использует такое же соотношение дисков с данными и контрольных дисков, как и RAID 3, но доступ к данным у них осуществляется по-разному. Информация о четности хранится в виде блоков и связана с набором блоков данных.

В RAID 3 каждое обращение направляется на все диски. Но некоторые приложения предпочитают небольшие обращения, допуская параллельное осуществление независимого доступа. В этом и заключается цель создания массивов RAID уровней от 4 до 6. Поскольку для определения корректности данных при чтении каждого сектора проверяется информация обнаружения ошибок, такие «небольшие считывания» с каждого диска могут происходить независимо, потому что минимально доступен один сектор. В среде RAID-массива небольшое обращение от носится только к одному диску из группы защиты, а большое обращение относится ко всем дискам этой группы.

При записи все обстоит по-другому. Небольшая запись данных потребует чтения старых данных и старого значения контроля четности с добавлением новой информации, а затем она потребует записи нового значения четности на избыточный диск и новых данных на диск с данными (рис. 4.13).

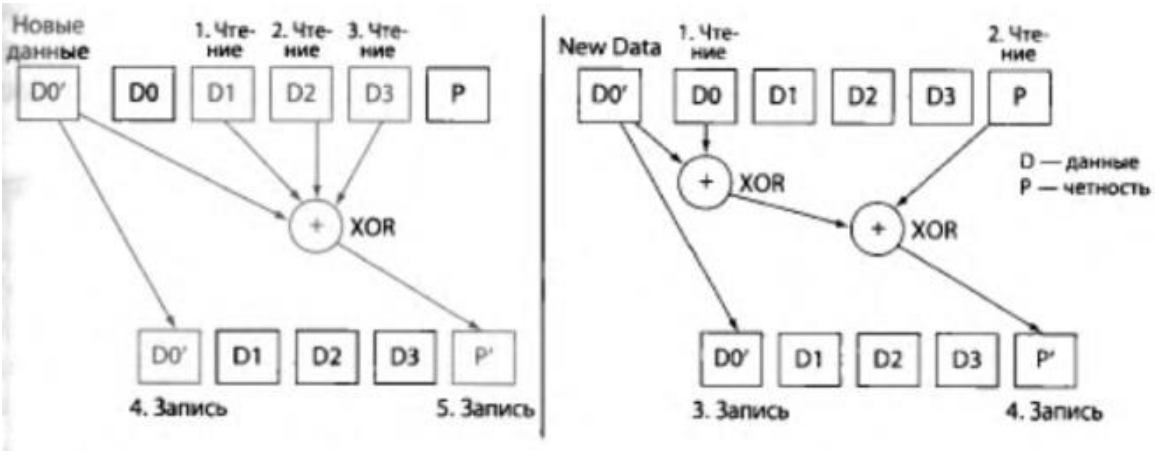


Рис. 4.13. Сравнение записи в RAID 3 (слева) и RAID 4 (справа)

Такая оптимизация (рис. 4.13) для небольших записей сокращает количество обращений к дискам, а также количество задействованных дисков. На рис. 4.13 предполагается, что используются четыре блока данных и один блок четности. Обычное вычисление четности в массиве RAID 3 в левой части рисунка требует чтения блоков D1. D2 и D3 перед добавлением блока DO' для вычислении нового значения четности P'. (Здесь нет ничего удивительного, новые данные DO' поступают прямо из центрального процессора, поэтому диски в их чтении не задействованы). Сокращение операций в массиве RAID 4. изображенное в правой части рисунка, предусматривает чтение старого значения DO, которое сравнивается с новым значением DO', чтобы узнать, какие биты изменились. Затем считывается старое значение четности P, после чего изменяются соответствующие биты для получения значения P'. В данном случае используется логическая функция исключающее ИЛИ (XOR).

В данном примере три чтения дисков (DI. D2, D3) и две записи на диски (DO'. P'), в которых были задействованы все диски, заменены двумя чтениями дисков (DO, P) и двумя записями на диски (DO', P'), в которых задействуются всего два диска. Увеличение размера группы четности влечет за собой экономию за счет сокращения числа операций. Такое же сокращение используется и в массиве **RAID 5.**

Ключевой догадкой, позволившей сократить издержки, явилось то, что четность – это просто сумма данных. Если отслеживать изменившиеся биты при записи новой информации, то потребуется изменить лишь соответствующие биты на диске четности. Сокращение количества операций показано в правой части рис. 4.13. Нужно прочитать старые данные с диска, на который будет вестись запись, сравнить их с новыми данными, чтобы узнать, какие биты изменились, прочитать старое значение четности, изменить соответствующие биты, а затем записать но-

вые данные и новое значение четности. Таким образом, небольшая запись повлечет за собой четыре обращения к двум дискам вместо обращения ко всем дискам.

Четность с распространенным чередованием (RAID 5)

В массиве RAID 4 эффективно поддерживаются разные сочетания больших чтений, больших записей и небольших чтений, плюс к этому имеется возможность небольших записей. Небольшим недостатком этой системы является необходимость обновления содержимого диска четности при каждой записи, поэтому диск четности становится узким местом для последовательных записей.

Чтобы устранить узкое место, связанное с записью значений четности, информация о четности может быть распространена по всем дискам, чтобы не было единого узкого места для записей. Организация распространенной информации о четности является особенностью массива RAID 5 (рис. 4.14). За счет распространения блоков четности по всем дискам некоторые небольшие записи могут вестись параллельно

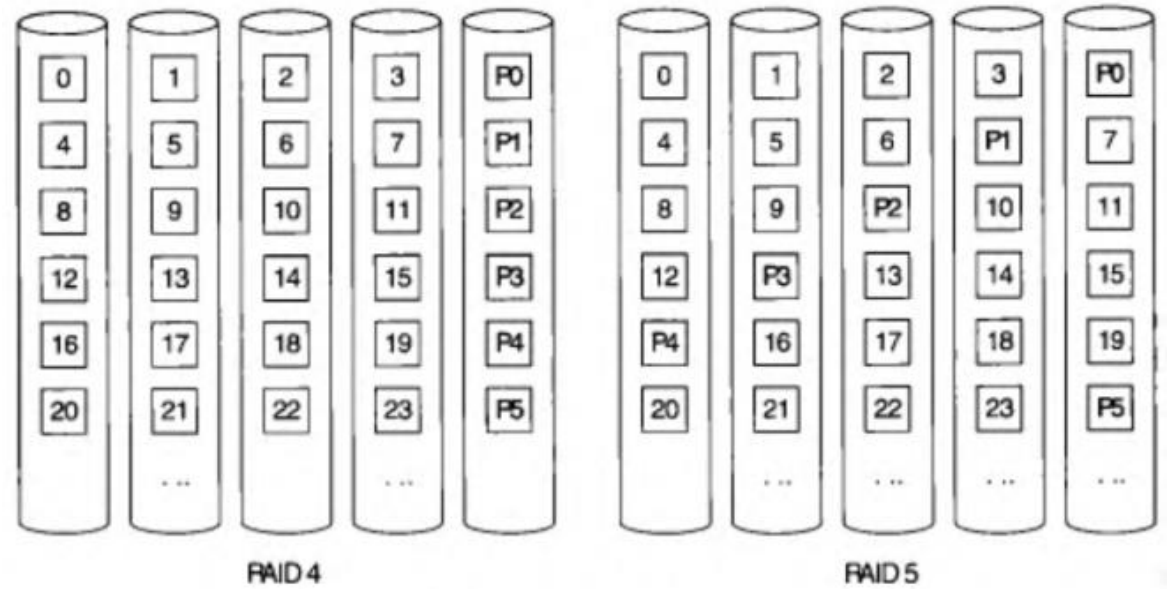


Рис. 4.14. Сравнение RAID 4 и RAID 5

На рис. 4.14 показано, как данные распространяются в массиве RAID 4 по сравнению с массивом RAID 5. Как показано в организации справа, в массиве RAID 5 четность связана с каждой строкой блоков данных и больше не ограничивается размещением на одном диске. Такая организация позволяет нескольким записям вестись одновременно, поскольку блоки четности больше не размещаются на одном и том же диске. Например, запись в блок 8 в правой части рисунка должна также

обращаться и к его блоку четности P2, задействуя таким образом первый и третий диски. Вторая запись в блок 5 в правой части рисунка предполагает обновление его блока четности P1, обращаясь ко второму и четвертому дискам, и поэтому она может вестись параллельно с записью в блок 8. Точно такие же записи при организации, показанной в левой части рисунка, приведут к изменениям в блоках P1 и P2, а оба этих блока расположены на пятом диске, что и составляет узкое место такой организации.

P+Q избыточность (RAID 6)

RAID 6 — похож на RAID 5, но имеет более высокую степень надёжности — три диска данных и два диска контроля чётности (рис. 4.15). Основан на кодах **Рида — Соломона** и обеспечивает работоспособность после одновременного выхода из строя любых двух дисков. Обычно использование RAID-6 вызывает примерно 10-15 % падение производительности дисковой группы, относительно RAID 5, что вызвано большим объёмом работы для контроллера (более сложный алгоритм расчёта контрольных сумм), а также необходимостью читать и перезаписывать больше дисковых блоков при записи каждого блока. Схемы, основанные на вычислении четности, защищают от единичного самоопределяющегося отказа. Когда проведения одной ликвидации последствий отказа недостаточно, четность может быть распространена, чтобы можно было иметь возможность второго вычисления данных и еще один диск с контрольной информацией. Второй контрольный блок позволяет восстановить информацию при втором отказе. Поэтому издержки хранения информации по сравнению с массивом RAID 5 удваиваются. Сокращение издержек небольшой записи, показанное на рис. 4.13, работает точно также, за исключением того, что теперь происходит шесть обращений к дискам вместо четырех для обновления информации как в P, так и в Q блоке.

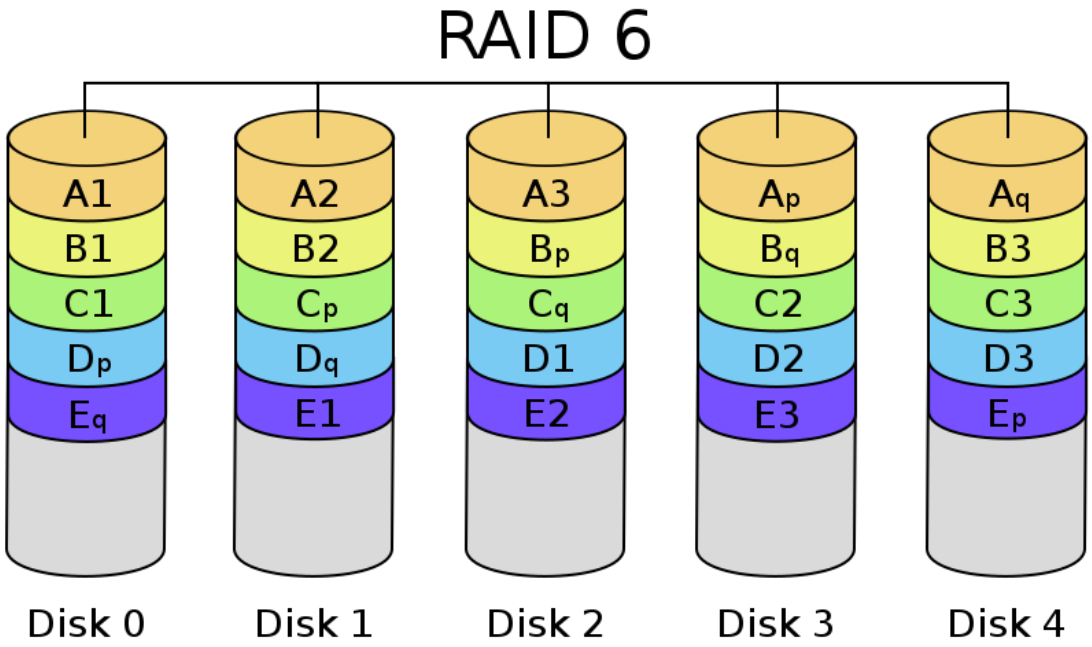


Рис. 4.15. RAID 6

Выводы по RAID-массивам

Массивы RAID 1 и RAID 5 широко используются на серверах; по одной из оценок 80% дисков на серверах имеют RAID-организацию.

Одним из слабых мест RAID-систем является ремонт. Во-первых, чтобы избежать недоступности данных во время ремонта, массив должен быть сконструирован так, чтобы отказавшие диски можно было заменять, не выключая системы. У RAID- массивов достаточно избыточности, чтобы позволить им работать в непрерывном режиме, но горячая замена дисков накладывает определенные требования на физическую и электрическую конструкцию массива и на интерфейсы, используемые при работе с дисками. Во-вторых, во время ремонта может произойти еще один отказ, поэтому время ремонта влияет на шансы потери данных: чем дольше ремонт, тем выше шансы еще одного отказа, который приведет к потере данных. Чтобы исключить вынужденное ожидание, пока оператор принесет исправный диск, в некоторые системы включаются ожидающие подключения резервы, чтобы данные могли быть тут же реконструированы, как только обнаружится отказ.

Помимо вопросов конструирования пригодной к ремонту RAID-системы, возникают вопросы о том, как как с течением времени изменяется технология дисковых устройств. Хотя производители дисков заявляют об очень высоких показателях MTTF для своих продуктов, все эти показатели соответствуют вполне определенным условиям эксплуата-

ции. Если отдельно взятый дисковый массив подвергся температурным перепадам, скажем, из-за поломки системы кондиционирования воздуха, или же он подвергся ударным нагрузкам из-за недостатков в конструкции или установки аппаратной стойки, интенсивность отказов может увеличиться в 3-6 раз. Вычисление готовности RAID-системы предполагает независимость отказов дисков, но эти отказы могут быть связаны друг с другом, поскольку повреждения из-за эксплуатационной среды вполне вероятно могут коснуться всех дисков массива.

Другая проблема возникает в связи с тем, что пропускная способность дисков растет намного медленнее, чем их емкость, время на ремонт диска в RAID-системе возрастает, что в свою очередь вызывает рост шансов на повторный отказ. Например, последовательное чтение SATA-диска емкостью 1000 Гбайт, если не будет никаких помех, может занять почти три часа. Учитывая то, что поврежденный RAID-массив, скорее всего, продолжает обслуживать данные, реконструкция может существенно затянуться. Кроме увеличения этого времени возникает также опасение, что чтение существенно большего объема данных в процессе реконструкции будет означать увеличение шансов на невосстанавливаемый отказ чтения носителя, который может повлечь за собой утрату данных. Еще одной проблемой, связанной с одновременно возникающими отказами, является рост числа дисков в массиве и использование SATA-дисков, работающих медленнее и имеющих более высокую емкость по сравнению с обычными серверными дисками.

Следовательно, потребителей привлекает защита сразу от нескольких отказов, и поэтому возрастают предложения массивов RAID 6 в качестве дополнений и в качестве основных устройств.

5 ФУНКЦИОНАЛЬНАЯ И СТРУКТУРНАЯ ОРГАНИЗАЦИЯ ГРАФИЧЕСКОГО ПРОЦЕССОРА

5.1 Введение в графические процессоры

Главным обоснованием добавления SIMD-инструкций к существующим архитектурам послужило то обстоятельство, что многие микропроцессоры в персональных компьютерах и рабочих станциях были подключены к дисплеям, поэтому на графику тратилось все больше вычислительного времени. Поэтому, когда согласно закону Мура количество доступных микропроцессорам транзисторов увеличилось, появился смысл улучшить обработку графики.

Большим стимулом для улучшения обработки графики стала индустрия компьютерных игр, развивавшаяся как на базе персональных компьютеров, так и на специальных игровых консолях, таких как Sony PlayStation, Xbox. Быстро разраставшийся рынок компьютерных игр подтолкнул многие компании на инвестирование все более крупных средств в развитие быстродействующего графического оборудования. Таким образом, темпы развития графической обработки стали более высокими по сравнению с темпами развития вычислений общего назначения в серийно выпускаемых микропроцессорах.

Сообщество разработчиков графических устройств и компьютерных игр имело несколько иные цели, чем сообщество разработчиков микропроцессоров, и выработало свой собственный стиль обработки данных и свою собственную терминологию. Как только графические процессоры стали мощнее, их начали называть **графическими процессорными устройствами (Graphics Processing Unit, GPU)**, чтобы отличать их от центральных процессорных устройств, CPU. Рассмотрим некоторые ключевые характеристики, благодаря которым графические процессоры отличаются от центральных процессоров:

- GPU являются **ускорителями**, дополняющими CPU, поэтому им не нужно уметь выполнять все задачи, присущие CPU. Эта роль позволяет им посвятить все свои ресурсы графике. Для GPU вполне нормально выполнять некоторые задачи недостаточно хорошо или вообще не выполнять, при условии, что в системе, где есть CPU и GPU, CPU может при необходимости эти задачи выполнить. Таким образом, комбинация CPU-GPU является одним

из примеров **гетерогенной мультипроцессорной обработки**, где не все процессоры являются идентичными.

- Интерфейсами программирования GPU являются **высокоуровневые интерфейсы прикладного программирования** (API), такие как OpenGL и разработанный компанией Microsoft DirectX, в совокупности с высокоуровневыми языками графического затенения (shading), такими как Cg, разработанный компанией NVIDIA язык C для графики, разработанный компанией Microsoft язык высокого уровня для программирования шейдеров High Level Shader Language (HLSL), OpenGL Shading Language (GLSL) и язык шейдеров от компании Apple – Metal. **Шейдер** – это программа для одной из ступеней графического конвейера, используемая в трёхмерной графике для определения окончательных параметров объекта или изображения. Компиляторы языков предназначены не для машинных инструкций, а для промежуточных языков, отвечающих промышленным стандартам. Программный драйвер GPU генерирует оптимизированные машинные инструкции, подходящие к конкретному GPU. Эти API и языки развиваются довольно быстро, охватывая все новые GPU-ресурсы. Отсутствие необходимости соблюдения обратной совместимости по двоичным инструкциям позволяет GPU-разработчиками исследовать новые архитектуры, не испытывая опасения за постоянные неудачи с реализацией. Такая среда способствует более быстрому появлению нововведений в GPU, чем в CPU.
- Обработка графики предполагает рисование точек (вершин) трехмерных геометрических примитивов, таких как линии и треугольники, затенение или прорисовку, или рендерниг (rendering), пиксельных фрагментов геометрических примитивов. В видеоиграх, к примеру, рисуется в 20-30 раз больше пикселей, чем вершин. Каждая вершина может быть нарисована независимо, как и прорисовка каждого пиксельного фрагмента, поэтому графические задачи отличаются большим уровнем параллелизма данных. Чтобы быстро прорисовать миллионы пикселей на кадр, создается GPU, способный параллельно выполнять множество потоков от программ, рисующих вершины и пиксельные затенения.

- Для достижения высокой производительности GPU полагаются на повсеместный параллелизм, обеспечиваемый множеством параллельных процессоров и множеством параллельных потоков.
- Графические процессоры полагаются на наличие достаточно большого количества потоков, позволяющее скрадывать латентность обращения к памяти. В то время как центральные процессоры преодолевают латентность при обращении к памяти главным образом за счет использования многоуровневой кэш-памяти. То есть между временем выдачи запроса к памяти и временем поступления данных GPU выполняет сотни или тысячи потоков, которые не зависят от этого запроса.
- В прошлом для достижения производительности, необходимой для графических приложений, графические процессоры были основаны на использовании разнотипных специализированных процессоров. Современные графические процессоры держат курс на применение одинаковых универсальных процессоров для предоставления большей гибкости в программировании, что делает их более похожими на многоядерные конструкции, наблюдаемые в вычислительной технике общего назначения.
- К типам графических данных относятся вершины, состоящие из координат (x,y,z,w), и пиксели, состоящие из цветов (красный, зеленый, синий, альфа). Графические процессоры представляют каждый компонент точки в виде 32-разрядного числа с плавающей точкой. Каждый из четырех компонентов пикселей сначала был 8-разрядным беззнаковым целым числом, но новые графические процессоры представляют каждый компонент в виде числа с плавающей точкой одинарной точности в диапазоне между 0,0 и 1,0.
- Учитывая четырехэлементную природу типов графических данных, графические процессоры исторически имели SIMD-инструкции, подобные инструкциям центральных процессоров. Современные GPU, чтобы облегчить программирование и повысить эффективность, больше сориентированы на скалярные инструкции.

- Рабочий набор графических данных может иметь объем в сотни мегабайт, и он не отличается такой же локальностью, связанной со временем, которая присуща данным в обычных приложениях. Оперативная память GPU в силу этих причин ориентирована на пропускную способность, а не на снижение времени латентности. Для GPU используются отдельные однотипные DRAM-модули с более высокими показателями разрядности и пропускной способности, чем у DRAM-модулей для центральных процессоров. Оперативная память графических процессоров традиционно имела меньший объем, чем оперативная память обычных микропроцессоров. В 2018 году у графических процессоров среднего ценового диапазона имеется не более 6 Гбайт оперативной памяти, в то время как у центральных процессоров той же ценовой категории это значение может быть от 8 до 32 Гбайт.
- Учитывая, что графические процессоры для достижения высокой полосы пропускания при обращениях к памяти зависят от использования большого количества потоков, в них может быть столько же параллельных процессоров, сколько и потоков. Следовательно, каждый графический процессор является в высокой степени многопоточным устройством.
- В отличие от центральных процессоров в GPU не было поддержки арифметики чисел с плавающей точкой двойной точности, поскольку для графических приложений она была не нужна. В 2008 году были представлены первые GPU с поддержкой чисел двойной точности. Тем не менее, даже на современных GPU операции с одинарной точностью по-прежнему осуществляются гораздо быстрее операций с двойной точностью, в то время как разница в производительности для центральных процессоров сводится количеству передач в системе памяти.

Хотя GPU были разработаны для более узкого круга приложений, некоторые программисты задавались вопросом, могли бы они придать своим приложениями такую форму, которая позволила бы им раскрыть высокую потенциальную производительность графических процессоров. Чтобы выделить этот стиль использования графических процессоров, специалисты назвали его универсальным использованием графиче-

ских процессоров (General Purpose GPUs, или **GPGPU**s). Следует иметь в виду, что для вычислений общего назначения нужно учитывать время на передачу данных между памятью CPU и памятью GPU, поскольку последний является сопроцессором.

После изнурительных попыток решить свои задачи с использованием графических API и языков программирования графических шейдеров, инженеры под влиянием языка C разработали языки, позволившие им создавать программы непосредственно для графических процессоров. В качестве примера можно привести следующие технологии:

- CUDA – технология GPGPU, позволяющая программистам реализовывать на языке программирования C (а также C++/C#) алгоритмы, выполнимые на графических процессорах ускорителей компании NVIDIA.
- OpenCL – фреймворк для написания компьютерных программ, связанных с параллельными вычислениями на различных графических и центральных процессорах, а также FPGA.
- DirectCompute – API для вычислений на GPU, предназначен для выполнения вычислений общего назначения на графических процессорах под управлением операционных систем семейства Microsoft Windows.
- Radeon Open Compute (ROCm) – программное обеспечение с поддержкой новых графических процессоров Radeon, математических библиотек и современных языков программирования для ускорения разработки высокопроизводительных и энергоэффективных гетерогенных вычислительных систем.
- OpenACC – программный стандарт для параллельного программирования, разрабатываемый совместно компаниями Cray, CAPS, NVIDIA и PGI. Стандарт описывает набор директив компилятора, предназначенных для упрощения создания гетерогенных параллельных программ, с использованием как центрального, так и графического процессоров.
- C++ AMP – библиотека, использующая DirectX 11, и открытая спецификация, созданные Microsoft для реализации параллельных программ для гибридных систем на языке C++.

С повышением программируемости стала расти и степень использования графических процессоров в параллельных вычислениях. Ви-

деокарты повсеместно используются в задачах машинного обучения, физических симуляциях, обработке сигналов, вычислительной математики/геометрии, операций с базами данных, вычислительной биологии, вычислительной экономики, компьютерного зрения и т.д. Для увеличения производительности в таких задачах в современных видеокартах имеются блоки работы с числами половинной точности и добавлена поддержка целочисленных операций формата int8. Для машинного обучения производители видеокарт вводят тензорные ядра (Tensor Core). Тензорные ядра специализируются на выполнении операций матричного перемножения, которые лежат в основе обучения и инференса нейронных сетей, они используются для умножения больших матриц входных данных и весов в связанных слоях сети.

5.2 Составные части видеокарты и её характеристики

Основой современной видеокарты является **графический процессор.** Он занимается расчетами выводимого изображения, освобождая от этой обязанности центральный процессор, производит расчеты для обработки команд трехмерной графики. Как и любой процессор, характеризуется **тактовой частотой и архитектурой.** Другие, не менее важные характеристики это количество выводимой информации в единицу времени – **текстурная и пиксельная скорости заполнения**, измеряются в млн. пикселей в секунду.

Графический процессор изготавливается по определенному техпроцессу. Напомним, что техпроцесс – технология изготовления основных микросхем видеокарты, и характеризуется характерным размером, измеряемым в нанометрах, современные микросхемы выпускаются по 28- или 14-нм нормам техпроцесса. Чем меньше данный параметр, тем больше элементов можно уместить на кристалле микросхемы.

Видеоконтроллер отвечает за формирование изображения в видеопамяти, осуществляет обработку запросов центрального процессора. Кроме этого, присутствуют контроллер внешней шины данных (например, PCI-e), контроллер внутренней шины данных и контроллер видеопамяти. Ширина внутренней шины и шины видеопамяти обычно больше, чем внешней (64-2048 разрядов против 32), для преобразования цифрового сигнала в аналоговый в видеоконтроллеры встраивается RAMDAC. Современные графические адаптеры обычно имеют не менее двух видеоконтроллеров, работающих независимо друг от друга и управляющих одновременно одним или несколькими дисплеями каждый.

Видеопамять – кадровый буфер, в котором хранится изображение, генерируемое и постоянно изменяемое графическим процессором и выводимое на экран монитора. В видеопамяти хранятся также промежуточные невидимые на экране элементы изображения и другие данные. Видеопамять бывает нескольких типов, различающихся по скорости доступа и рабочей частоте. Современные видеокарты комплектуются памятью типа GDDR5, GDDR5X, HBM2. **Частота видеопамяти** отлична от частоты ядра видеокарты, как и в случае центрального процессора и оперативной памяти. Следует также иметь в виду, что помимо видеопамяти, находящейся на видеокарте, современные графические процессоры обычно используют в своей работе часть общей системной памяти компьютера, прямой доступ к которой организуется драйвером видеоадаптера через шину PCI-E. **Ширина шины памяти** – количество бит информации, передаваемой за такт. **Объем видеопамяти** – объем встроенной оперативной памяти видеокарты, измеряется в мегабайтах.

Видео-ПЗУ – постоянное запоминающее устройство, в которое записаны видео-BIOS, экранные шрифты, служебные таблицы и т.п. ПЗУ не используется видеоконтроллером напрямую – к нему обращается только центральный процессор. Хранящийся в ПЗУ видео-BIOS обеспечивает инициализацию и работу видеокарты до загрузки основной операционной системы, а также содержит системные данные, которые могут читаться и интерпретироваться видеодрайвером в процессе работы. Видео-BIOS содержит основные команды, которые предоставляют интерфейс между оборудованием видеоадаптера и программным обеспечением. На многих современных картах устанавливаются электрически перепрограммируемые ПЗУ (EEPROM, Flash ROM), допускающие перезапись **Flash BIOS** самим пользователем при помощи специальной программы.

Система охлаждения предназначена для сохранения температурного режима видеопроцессора и видеопамяти в допустимых пределах. Существует пассивное охлаждение (радиатор, тепловые трубки) и активное охлаждение (кулеры, системы жидкостного охлаждения, системы охлаждения на элементах Пельтье, системы фазового перехода, системы экстремального охлаждения с жидким азотом в виде хладагента).

Интерфейсы передачи данных. Первоначально видеоадаптер имел всего один разъем VGA. В настоящее время платы оснащают одним или несколькими разъемами DVI, HDMI, Display Port. Dispay Port позволяет подключать до четырех устройств, в том числе акустические системы, USB-концентраторы и иные устройства ввода-вывода. На ви-

деокарте также возможно размещение композитных и S-Video видеовыходов и видеовходов (обозначаются, как ViVo).

В настоящий момент все видеокарты используют расширение шины PCI-E (PCI Express) версии 3.0, это последовательный, в отличие от AGP, интерфейс, его пропускная способность может достигать 16 Гбайт/с. Но в начале 90х годов для видеоадаптеров в качестве интерфейса использовалась шина PCI (Periferal Component Interconnect). С появлением 3D-игр со сложной графикой, пропускной способности PCI стало недостаточно. Фирма Intel модернизировала шину PCI, обеспечила отдельный доступ к памяти с поддержкой некоторых специфических запросов видеоадаптеров, и назвала это AGP (Accelerated Graphics Port).

Правильная и полнофункциональная работа современного графического адаптера обеспечивается с помощью **видеодрайвера** – специального программного обеспечения, поставляемого производителем видеокарты и загружаемого в процессе запуска операционной системы. Видеодрайвер выполняет функции интерфейса между системой с запущенными в ней приложениями и видеоадаптером.

5.3 Введение в архитектуру графического процессора NVIDIA

Поскольку графические процессоры развивались в своей собственной среде, у них не только другая архитектура, как было сказано выше, но и другой набор терминов. По мере изучения терминологии графических процессоров станет ясно сходство с подходами, рассмотренными в предыдущих разделах, например, сходство с мелкомодульной многопоточностью и векторами.

Чтобы помочь с переходом к новым словам, будет представлено краткое введение в термины и идеи, используемые в архитектуре NVIDIA Pascal и в программной среде CUDA.

Отдельная микросхема GPU расположена на карте, которая подключена к персональному компьютеру через разъем шины PCI-Express (**дискретная видеокарта**). Так же GPU могут быть встроены в материнскую плату, интегрированы в чипсет материнской платы, например, в северный мост или в южный мост. Некоторые процессоры содержат видеоядра (**интегрированная видеокарта**). Интегрированные видеокарты используют оперативную память компьютера в качестве ОЗУ. Можно встретить **гибридные решения**, например, Optimus от NVIDIA и DualGraphics от AMD, которые позволяют использовать одновременно дискретную и интегрированную видеокарты. Дискретные видеокар-

ты могут быть внешними (**eGPU**), то есть находиться вне корпуса компьютера.

Графические процессоры обычно предлагаются в виде семейства микросхем с разными диапазонами стоимости – производительности, сохраняющими программную совместимость друг с другом. Наборы микросхем на основе NVIDIA Pascal предлагаются в диапазоне от 3 до 30 узлов, которые NVIDIA называет **мультипроцессорами (streaming multiprocessor, SM)**. В начале 2018 года самая старшая версия называлась TITAN Xp, имела 30 мультипроцессоров и тактовую частоту 1,5 ГГц, не учитывая Tesla P100, применение которого ограничено сферой суперкомпьютеров в силу специального форм-фактора. Каждый мультипроцессор делится на четыре **блока обработки.** Мультипроцессор имеет следующие **многопоточные скалярные блоки**: 128 ядер CUDA для работы с числами с плавающей точкой одинарной точности (FP32), 4 ядер CUDA двойной точности (FP64), и одно ядро CUDA для работы с двумя числами FP16x2 (рисунок 5.1).

Поскольку архитектура включает инструкции умножения-сложения чисел с плавающей точкой одинарной точности, пиковая производительность умножения-сложения одинарной точности чипсета TITAN Xp равна:

$$30\ SM\ \times \frac{128\ FP32}{SM} \times \frac{2\ \text{Флоп/инстр}}{FP32} \times \frac{1\ \text{инстр}}{\text{такт}} \times \frac{1{,}5 \times 10^9\ \text{тактов}}{\text{в секунду}}$$
$$= 11520\ \text{Гфлоп/сек}$$

Производитель приводит значение 11366,4 Гфлоп/сек.

Каждый из 30 мультипроцессоров в составе TITAN Xp имеет локальное программно-управляемое запоминающее устройство объемом 48 Кбайт (**кэш L1**), четыре **регистровых файла** размеров 64 Кбайт (16384x4 32-х разрядных регистра), так как мультипроцессор разделен на 4 эквивалентные части, а так же общую память размером 96 Кбайт.

Каждый из мультипроцессоров SM спарен с PolyMorph Engine, который в разных стадиях работы обрабатывает текстурные выборки, тесселяцию, трансформацию, установку вершинных атрибутов и коррекцию перспективы, считает несколько проекций геометрии благодаря технологии мультипроецирования Simultaneous Multi-Projection (все перечисленные технологии будут рассмотрены ниже). Результаты, вычисленные в каждой стадии, передаются в мультипроцессор SM. Последний выполняет шейдерную программу, возвращая данные к следующей стадии PolyMorph Engine. Комбинация мультипроцессора SM с одним

движком Polymorph Engine традиционно для NVIDIA называется TPC — Texture Processor Cluster (рисунок 5.2).

Рис. 5.1. Архитектура мультипроцессора NVIDIA Pascal

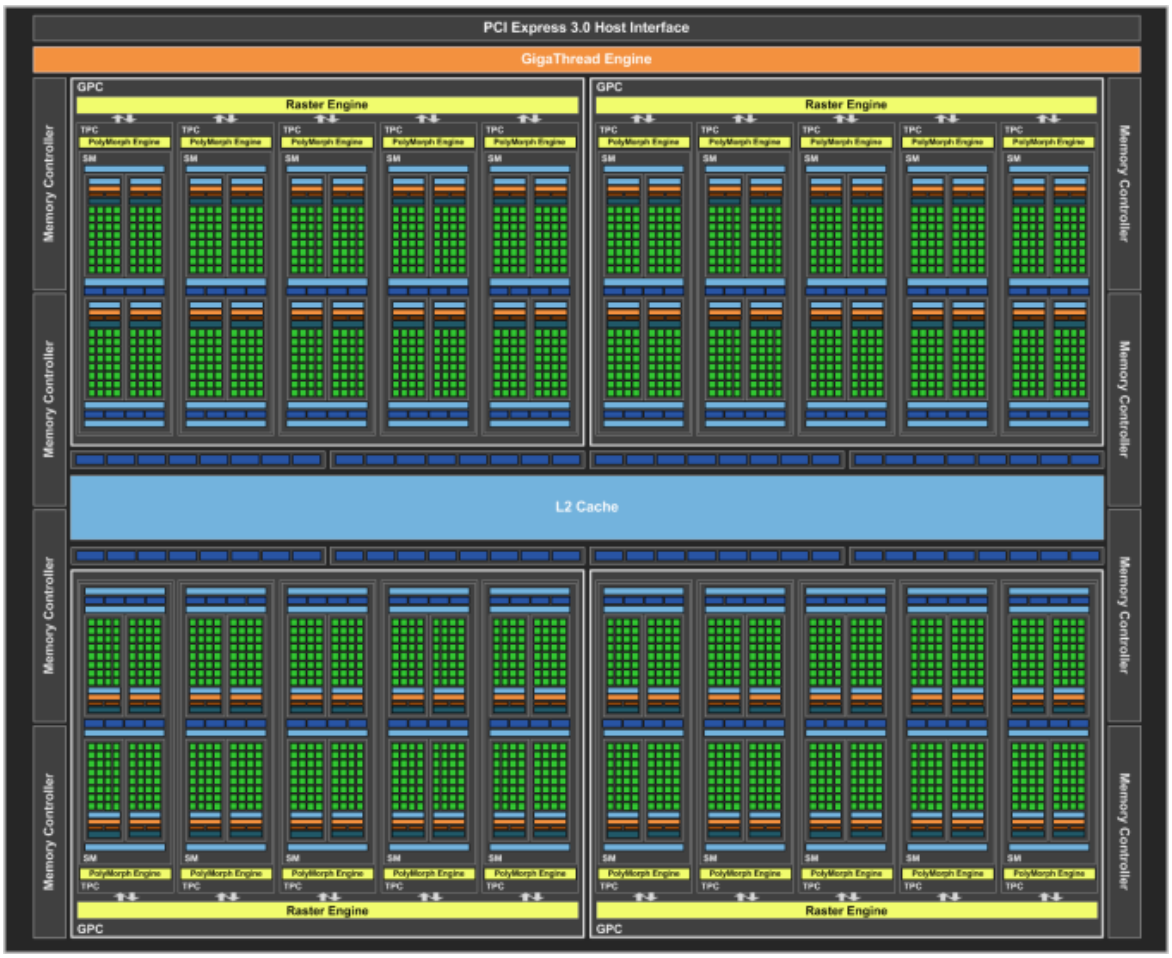


Рис. 5.2. Архитектура NVIDIA Pascal

Каждые 5 мультипроцессоров объединены в кластеры обработки графики (GPC) с общим движком растеризации Raster Engine. Данные в Raster Engine поступают после прохождения всех стадий PolyMorph Engine. Движок растеризации выполняет три стадии конвейера: выборка положения вершин и вычисление проекции граней треугольника, вычисление покрытия пикселей, отбрасывания невидимых поверхностей.

Всего в чипе 96 блоков растеризации ROP, L2 кэш объемом 3072 Кбайт, 240 текстурных блоков (TMU) по 8 в каждом SM. Назначение ROP и TMU будет раскрыто позднее.

Система памяти TITAN Xp состоит из 12 сегментов 1426 МГц графической GDDR5X DRAM-памяти, каждый шириной 32 бит и объемом 1024 Мбайт. Общий объем памяти, таким образом, составляет 12288 Мбайт.

При расчете пиковой пропускной способности памяти учтем, что GDDR5X и GDDR5 имеют две физические тактовые частоты: **дифференциальная тактовая частота CK** (для передачи адресов и команд) и

дифференциальная тактовая частота WCK (для передачи данных). WCK всегда функционирует на тактовой частоте, вдвое превышающей таковую у CK. Память GDDR5 работает в режиме DDR (double data rate) и может передавать два бита данных на каждый синхроимпульс WCK как по фронту, так и по спаду тактового импульса. Память GDDR5X поддерживает новый режим работы **QDR** (quad data rate) и может передавать до четырёх бит данных на каждый синхроимпульс WCK. Таким образом, для расчета пиковой пропускной способности памяти GDDR5X и GDDR5 частоту будет необходимо увеличить в 2 раза. Для памяти GDDR5X за один так передается 4 бита. Тогда пиковая пропускная способность памяти GDDR5X равна

$$12 \times \frac{4\text{ байт}}{\text{на передачу}} \times \frac{4\text{ передачи}}{\text{за такт}} \times \frac{2 \times 1{,}4 \times 10^9 \text{тактов}}{\text{в секунду}} = 537\text{ Гбайт/сек}$$

Другой пример, для видеокарты NVIDIA 1060 6 Гб: память состоит из 6 сегментов 2002 МГц графической GDDR5 DRAM-памяти, каждый шириной 32 бит и объемом 1024 Мбайт. Пиковая пропускная способность памяти:

$$6 \times \frac{4\text{ байт}}{\text{на передачу}} \times \frac{2\text{ передачи}}{\text{за такт}} \times \frac{2 \times 2 \times 10^9 \text{тактов}}{\text{в секунду}} = 192\text{ Гбайт/сек}$$

Чтобы скрыть латентность памяти, каждый потоковый процессор имеет аппаратно-поддерживаемые потоки. За это отвечает планировщики потоков и декодеры (dispatch unit), а так же алгоритмы управления потоками и **варпами** (warp). Каждая группа из 32 потоков называется варпом. Варп является блоком диспетчеризации, и активные потоки в варпе, до 64, выполняются в параллельном режиме SIMD-способом. Но многопоточная система справляется с условиями, позволяя потокам расходиться по разным путям условных переходов. Когда потоки варпа идут по расходящимся путям, варп последовательно выполняет код по обоим маршрутам, делая неактивными некоторые потоки, что приводит к более медленному выполнению активных потоков. Как только условная часть завершится, оборудование снова объединяет потоки в полностью активный варп. Для достижения наивысшей производительности все 32 потока варпа нуждаются в совместном параллельном выполнении. Похожим образом оборудование также следит за течением адресов, поступающих от разных потоков, чтобы попытаться объединить отдельные запросы в меньшее количество передач более крупных блоков памяти для увеличения производительности при работе с памятью.

Как ранее уже упоминалось, наивысшая производительность достигается, когда все 32 принадлежащих варпу потока выполняются вместе в SIMD-подобной манере, что в архитектуре Pascal называется «одна инструкция –несколько потоков» (**single-instruction multiple-thread, SIMT**). SIMT в динамическом режиме обнаруживает, какие принадлежащие варпу потоки могут выполнять вместе одну и ту же инструкцию и какие независимые потоки будут простаивать в данном цикле.

У среды программирования CUDA также есть своя собственная терминология. Программа CUDA является унифицированной C/C++ программой, предназначенной для гетерогенного центрального процессора и системы GPU. Она выполняется на центральном процессоре и осуществляет диспетчеризацию параллельной работы для GPU. Эта работа состоит из передачи данных от оперативной памяти и из диспетчеризации потоков. Поток является частью программы для GPU. Программисты указывают количество потоков в поточном блоке и количество поточных блоков, которое требуется запустить на выполнение на GPU. Программисты заботятся о поточных блоках, поскольку все потоки в блоке подвергаются диспетчеризации для запуска на одном и том же мультипроцессоре, поэтому они совместно используют одну и ту же локальную память. Благодаря этому они могут обмениваться данными через загрузки и сохранения, а не через сообщения. Компилятор CUDA распределяет регистры каждому потоку, при том ограничении, что количество регистров на один поток (максимум 255), умноженное на количество потоков на поточный блок, не должно превышать число, равное 65536 регистрам на один блок обработки мультипроцессора.

Мультипроцессор может иметь до 1024 потоков. Каждая группа из 32 потоков в потоковом блоке запакована в варпы. Большие потоковые блоки более эффективны, чем небольшие, и они могут уменьшаться до одного блока. Как уже ранее упоминалось, потоковые блоки и варпы, состоящие менее чем из 32 потоков, работают менее эффективно, чем заполненные.

Аппаратная диспетчеризация старается, по возможности, спланировать выполнение нескольких потоковых блоков на мультипроцессоре. Если это удается, диспетчер также динамически делит объем локального запоминающего устройства между различными потоковыми блоками.

5.4 Классификация графических процессоров

Современные графические процессоры, подобные рассмотренному выше, относятся к MIMD. Но остается открытым вопрос: как классифи-

цировать каждый блок мультипроцессоров Pascal и его скалярные процессоры?

Ранее утверждалось, что SIMD больше всего подходит для циклов for и меньше всего подходит для операторов case и switch. Pascal стремится к достижению высокой эффективности при параллелизме на уровне данных, облегчая программистам работу с независимым параллелизмом на уровне потоков. Pascal позволяет программисту считать мультипроцессор многопоточным MIMD-устройством, состоящим из 128 скалярных процессоров, но оборудование старается собрать вместе 32 скалярных процессоров для работы в качестве SIMT-устройства, когда несколько потоков одного и того же варпа могут выполняться вместе. Когда потоки работают независимо друг от друга и следуют по независимым путям выполнения, они выполняются медленнее, чем в режиме SIMT, поскольку все 32 потока в варпе совместно используют один блок извлечения инструкций.

Таким образом, у каждого независимого потока есть свой собственный эффективный персональный компьютер, поэтому программисты могут представлять мультипроцессор Pascal в виде MIMD-устройства, но, чтобы добиться желаемой производительности, они должны позаботиться о написании инструкций потока управления, позволяющих SIMT-оборудованию выполнять CUDA-программы в SIMD-режиме.

В таблице 5.1 приведена классификация, рассматривающая параллелизм на уровне инструкций в противоположность параллелизму на уровне данных, и показывается, где обнаруживается параллелизм – во время компиляции или в процессе выполнения. Эта классификация свидетельствует о том, что графические процессоры Pascal являются новым словом в компьютерной архитектуре.

Таблица 5.1 – Аппаратная классификация процессорных архитектур и примеров, основанная на статическом способе в противоположность динамическому и на параллелизме на уровне инструкций в противоположность параллелизму на уровне данных

	Статический способ: обнаруживается во время компиляции	Динамический способ: обнаруживается в процессе выполнения
Параллелизм на уровне инструкций	VLIW	Суперскаляр
Параллелизм на уровне	SIMD или вектор	Мультипроцессор Pascal

	Статический способ: обнаруживается во время компиляции	Динамический способ: обнаруживается в процессе выполнения
данных		

В отличие от векторной архитектуры, которая для распознавания параллелизма на уровне данных и генерации векторных инструкций полагается на векторизирующий компилятор, аппаратная реализация архитектуры Pascal обнаруживает параллелизм на уровне данных среди потоков в процессе выполнения. Таким образом, графические процессоры Pascal не нуждаются в векторизирующих компиляторах, и они облегчают программисту работу с частями программы, не имеющих параллелизма на уровне данных.

5.5 Обработка изображения в GPU

Для создания анимированной последовательности трехмерных изображений компьютеру необходимо математически анимировать последовательность кадров между ключевыми позициями. У прыгающего мяча, например, есть три ключевые позиции: подскок вверх, падение вниз и соприкосновение с поверхностью. Используя эти позиции в качестве шаблона, компьютер создает промежуточное изображение между разными позициями перемещения мяча, в результате его движение будет отображаться самым естественным образом. Кроме этого для отображения мяча на мониторе необходимо провести геометризацию – определить размеры, ориентации и расположения примитивов в пространстве и рассчитать влияния источников света. Примитивы – это простые геометрические объекты, с помощью которых конструируются более сложные объекты.

После геометризации выполняется функция растеризации – преобразование примитивов в пиксели на экране с нанесением нужных затенений и текстур. **Блоки растеризации** (ROP) осуществляют операции записи рассчитанных видеокартой пикселей в буферы кадра видеокарты и операции их смешивания, выполняют сглаживание, применяют фильтры постобработки.

Для окрашивания изображения можно использовать плоскостное затенение, при котором объект заполняется однородным цветом, либо затенение Гуро, которое позволяет присвоить цвет определенным точкам формы, но гораздо более эффективный метод и требовательный – это **наложение текстур**. Трехмерная программа накладывает двухмер-

ные изображения или поверхности на примитивы. Их положение задается расположением определяющих точек, обычно вершин.

Трехмерная программа имеет возможность изменять внешний вид каждой текстуры путем перспективной коррекции и затенения для получения эффекта трехмерности. В некоторых приложениях используется другая процедура, называемая множественным наложением (**mip mapping, MIP**); в этом случае применяются различные версии одной и той же текстуры, которые содержат разное количество деталей в зависимости от расстояния до объекта в трехмерном пространстве. При отображении удаляющихся объектов уменьшается насыщенность и яркость цветов текстуры.

Блоки текстурирования (TMU) работают совместно с вычислительными процессорами, ими осуществляется выборка и фильтрация текстурных и прочих данных, необходимых для построения сцены и универсальных вычислений. Хотя в последнее время больший упор делается на математические расчеты, а часть текстур заменяется процедурными, нагрузка на блоки TMU и сейчас довольно велика, так как кроме основных текстур, выборки необходимо делать и из карт нормалей и смещений, а также внеэкранных буферов рендеринга. Особенное влияние этот параметр оказывает на скорость рендеринга изображения при использовании анизотропной фильтрации, требующего дополнительных текстурных выборок, а также при сложных алгоритмах мягких теней.

Важным этапом визуализации являются шейдеры. Графические архитектуры стремятся к унификации поэтому, если раньше пиксельные шейдеры выполняли блоки пиксельных шейдеров, а вершинные – вершинные блоки, то с некоторого времени производители используют универсальные блоки, которые могут заниматься различными расчётами: вершинными, пиксельными, геометрическими и даже универсальными вычислениями.

Конвейер обработки состоит из трех основных шейдеров, но так же в современных устройствах могут использоваться и дополнительные.

Вершинный шейдер (vertex shader) оперирует данными, сопоставленными с вершинами многогранников. К таким данным, в частности, относятся координаты вершины в пространстве, текстурные координаты, тангенс-вектор, вектор бинормали, вектор нормали. Вершинный шейдер может быть использован для видового и перспективного преобразования вершин, генерации текстурных координат, расчета освещения и т. д. Примеры того, для чего применяются вершинные шейдеры: ски-

нинг для скелетной анимации, деформация объектов, волны, анимация объектов, имитация ткани.

Геометрический шейдер (geometry shader), в отличие от вершинного, способен обработать не только одну вершину, но и целый примитив. Это может быть отрезок (две вершины) и треугольник (три вершины), а при наличии информации о смежных вершинах может быть обработано до шести вершин для треугольного примитива. Пример обработки изображения геометрическим шейдером – построение травы, каскадные тени, billboarding. Billboarding – метод построения удаленных высокополигональных объектов в виде многократно размноженных четырехугольников, все копии которых всегда повернуты к камере. Примерами могут служить массив зданий, лес.

Пиксельный шейдер (pixel shader) работает с фрагментами изображения, под которыми в данном случае подразумеваются пиксели, обладающие некоторым набором атрибутов, таких как цвет, глубина, текстурные координаты. Пиксельный шейдер используется на последней стадии графического конвейера для формирования фрагмента изображения. Перед выполнением пиксельного шейдера данные обрабатываются в блоках растеризации. Большая часть игр сейчас ограничена производительностью исполнения пиксельных шейдеров. Примеры обработки изображения пиксельным шейдером: мультитекстурирование, попиксельное освещение, процедурные текстуры (мелкие детали лица, дерево), постобработка кадра (размытие, смазывание в движении, глубина резкости), пыль.

5.6 Технологии трехмерной графики

В различных видеоадаптерах применяются разные технологии визуализации. В настоящее время практически во всех видеоадаптерах фильтрация и основная визуализация выполняются за один проход, что позволяет увеличить частоту кадров. Видеоадаптеры с функцией **однопроходной визуализации и фильтрации** обычно являются более быстродействующими при работе с трехмерными программами и позволяют избежать искажений, вызванных ошибками в множественных вычислениях значений с плавающей запятой во время визуализации.

Новым словом является технология **однопроходного стереорендеринга**. Однопроходный стереорендеринг используется в VR. В случае использования этой возможности, графический процессор обрабатывает геометрию сцены лишь один раз, генерируя сразу две проекции для

каждого глаза, что вдвое снижает геометрическую нагрузку на GPU, а также снижает потери от работы драйвера и ОС.

В добавлению к этому используется отрисовка с разным разрешением (Multi-Res Shading). При ее применении, используется более высокое разрешение в центре кадра, а на периферии оно снижается для получения более высокой скорости рендеринга. Еще более продвинутым методом повышения эффективности VR-рендеринга является Lens Matched Shading, когда при помощи нескольких проекций имитируются геометрические искажения, требуемые при VR-рендеринге. Этот метод использует мультипроецирование для рендеринга 3D-сцены на поверхность, которая приближенно похожа на скорректированную линзой VR-шлема, что позволяет не отрисовывать много лишних пикселей на периферии, которые будут отброшены.

Ниже перечислены технологии трехмерной графики, наиболее часто используемые в современных GPU.

Затуманивание. Имитация газа или тумана в играх.

Альфа-смешивание. Одна из первых технологий трехмерной графики, используемая для создания реалистичных объектов, например «прозрачного» дыма, воды и стекла.

Z-буферизация. Часть видеопамяти, отведенная под Z-буфер, содержит информацию о глубине сцены. При визуализации эта информация служит для построения законченного изображения. Пиксели, которые располагаются ближе, будут визуализированы, в отличие от пикселей, закрытых другими объектами.

Рельефное текстурирование. Parallax Mapping – это техника аппроксимации эффекта смещения точек поверхности в зависимости от изменения точки зрения. Parallax mapping не создает 3D-объектов в обычном понимании этого слова. Например, пол или стена в игровой сцене будут выглядеть шероховатыми, оставаясь на самом деле абсолютно плоскими. Эффект рельефности здесь достигается лишь за счет манипуляций с текстурами. Метод искажает наложение текстуры, изменяя текстурные координаты так, что когда камера направлена на поверхность под разными углами, поверхность выглядит выпуклой. Две предшествующие техники рельефного текстурирования: Bump Mapping и Normal Mapping.

Наложение карт смещения (Displacement Mapping) является методом добавления деталей к трехмерным объектам. В отличие от рельефного текстурирования, когда картами высот правильно моделируется только освещенность точки, но не изменяется ее положение в пространстве, что дает лишь иллюзию увеличения сложности поверхности, кар-

ты смещения позволяют получить настоящие сложные 3D объекты из вершин и полигонов. В этом методе используется **тесслеяция**: в GPU посылается несложная геометрически модель отрисовываемого объекта, а аппаратный тесселятор разбивает её на большее количество геометрических примитивов, необходимое для текущей сцены. Затем эти вершины смещаются на необходимое расстояние для добавления детализации. Но сама по себе тесселяция не добавляет деталей, а только сглаживает модель. Поэтому к ней нужно ещё применить карту смещения высот. Исходная модель геометрически простая, что означает высокую эффективность хранения и передачи данных. Например, анимация рассчитывается для простой исходной модели, что делает возможным применение более сложных анимационных алгоритмов. Другим важным преимуществом является возможность гибкого изменения результирующей геометрической сложности, или использование динамического уровня детализации (LOD). Так же можно обойтись одной и той же моделью и картой смещения для разных игровых платформ, задавая уровень детализации при помощи разбиения на различное количество примитивов.

Динамическое суперразрешение (NVIDIA DSR или AMD VSR). Динамическое суперразрешение позволяет рассчитывать кадр в играх в более высоком разрешении, а затем масштабирует полученный результат до разрешения монитора.

Интегрированные функции трансформации объектов и распределения освещения (T&L). При формировании трехмерной анимации объект трансформируется при переходе из одного кадра в другой, после чего освещение изменяется в соответствии с перемещением объекта.

Воксельная глобальная иллюминация (NVIDIA VXGI) – реалистичное освещение, затенение и отражения в играх. Эта технология глобального освещения использует воксельную сетку для захвата информации об освещении в каждой точке сцены, а трассировка сцены осуществляется конусами для расчета эффекта отраженного света.

NVIDIA PhysX (AMD Accelerated Parallel Processing). Изначально за игровую физику целиком отвечал центральный процессор, но в силу того, что для обработки сцен реализма в основном нужны вычисления с плавающей запятой, то CPU, зачастую, становился узким горлышком в системе, поэтому данные вычисления перешли на GPU. Операции с вещественными числами всегда были слабым местом центрального процессора, производительность графического на порядки выше.

NVIDIA hairworks (AMD TressFX Hair) – система, моделирующая физические свойства волос в реальном времени. Обычно шерсть и воло-

сы в играх создаются посредством добавления к изображению персонажей полигональных линий и прозрачных текстур, с этой технологией добавляются сотни тысяч динамичных и реалистичных волосков к изображениям персонажей, существ и игровых миров.

AMD Partially Resident Textures. Виртуальные текстуры разбиваются на части фиксированного размера 64 Кбайта. В локальную память видеокарты подгружаются только те из них, которые нужны при рендеринге текущего кадра. Технология предполагает использование MIP-текстур. Когда шейдерная программа запрашивает данные, то некоторые данные уже есть в памяти (попадание), а других нет (промах). При промахе программа может запросить данные из MIP-текстуры меньшего разрешения, тогда изображение будет нечётким, но оно отрисуется без задержки. А к рендерингу следующего кадра оно уже может быть подгружено в кэш.

Тайловый рендеринг (Tile Based Rendering, TBR) – концепция является противоположностью рендерингу **Immediate Mode Rendering** (IMR). Процесс растеризации IMR выполняется для всего кадра, в случае же TBR кадр разделяется на множество прямоугольных тайлов, и процесс растеризации выполняется уже отдельно для тайлов. Как правило, тайлы имеют размер 16x16 или 32x32 пикселей.

Рендеринг в широком динамическом диапазоне (High Dynamic Range, HDR) добавляет больше градаций цвета в кадр и позволяет детализировать сцену. Суть HDR заключается в описании интенсивности и цвета реальными физическими величинами или линейно пропорциональными. Для описания используются не целые числа, а числа с плавающей точкой с большой точностью. Самый распространенный пример – это изображение затемненного помещения с окном на яркую улицу в солнечный день. С RGB моделью (целые числа) можно получить или нормальное отображение того, что находится за окном, или только того, что находится внутри помещения.

Тройной буфер для вывода кадров, обеспечивает одновременно сверхнизкие задержки в отрисовке и обеспечение вертикальной синхронизации. В одном буфере хранится выводимое изображение, в другом – последний отрисованный кадр, в третьем – рисуется текущий.

Фильтрация текстур бывает точечная, билинейная, трилинейная и анизатропная. Если объект находится параллельно плоскости экрана, то его одному пикселю соответствует один тексель (**тексель** – это пиксель двухмерного изображения, наложенного на 3D-поверхность). Когда объект наклонен или находится вдали от плоскости экрана, на один пиксель приходится несколько текселей, и поскольку монитор имеет

ограниченное количество пикселей, то цвет каждого приходится рассчитывать из нескольких текселей путем фильтрации.

- **Точёная фильтрация.** Цвет отображаемого пикселя соответствует цвету текселя, который находится ближе всего к центру светового пятна от пикселя.
- **Билинейная фильтрация**. Цвет отображаемого пикселя соответствует среднему цвету от четырех текселей, ближайшие к центру пикселя.
- **Трилинейная фильтрация**. Комбинация билинейной фильтрации и mip mapping.
- **Анизотропная фильтрация.** Принцип работы заключается в том, что берется MIP-текстура, установленная поперёк направления обзора, после чего происходит усреднение значений ее цветов с цветом некого количества текселей вдоль направления обзора. Количество текселей варьируется от 16 (для x2 фильтрации) до 128 (для x16). По сути, вместо квадратного фильтра, как в билинейной фильтрации, используется вытянутый, что позволяет более качественно выбрать нужный цвет для экранного пикселя.

Уменьшение неровностей, возникающих при увеличении разрешения, посредством сглаживания цветовых границ для обеспечения плавных цветовых переходов называется **полноэкранное сглаживание** (Anti-Aliasing, AA). Так же можно встретить такое определение: способ устранения «ступенчатости» на краях объектов, линий, которые находятся под наклоном и не являются ни строго вертикальными и ни строго горизонтальными. Очень интересно наблюдать за развитием полноэкранного сглаживания на протяжении нескольких лет.

На данный момент есть два основных подхода к сглаживанию: **мульти-выборка** (multi-sampling) и **пост-обработка**. Но при этом так же вводится гибридная мультивыборка, постобработка и техники временной фильтрации. Методы мульти-выборки обрабатывают дополнительные пиксели (больше разрешения дисплея). Количество дополнительных пикселей, как правило, выражается в виде коэффициента. Например, можно увидеть значение «4x MSAA», значит, часть сглаживаемого кадра будет генерироваться в 4x-кратном разрешении. Чем выше коэффициент, тем выше качество, но также и сильнее влияние на работу видеопамяти и частоту кадров. Методы класса пост-обработки не полагаются на расширенное разрешение, и таких понятий, как 2xFXAA

или 4xMLAA, не существует. В данном случае, сглаживание либо включено, либо выключено.

Методы сглаживания мульти-выборка:

- Суперсэмплинг (**SSAA**) – это первый метод сглаживания, который появился в игровых видеокартах и который является наиболее качественным, но весьма требовательным к ресурсам. SSAA – технология сглаживания, когда при четырехкратном сглаживании видеокарта готовит кадр в разрешении вчетверо выше, чем выводит на экран, потом происходит усреднение цвета соседних пикселей и вывод на экран в исходном разрешении.
- На смену SSAA пришёл мультисэмплинг (**MSAA**), более быстрый метод, но имеющий свои недостатки в плане качества. MSAA (Multisample anti-aliasing) – улучшенная версия SSAA, которая потребляет гораздо меньше ресурсов. Принцип заключается в том, что нецелесообразно то, что находится внутри текстуры, если «лесенки» есть только на краях. При качестве SSAA работает быстрее в 2 раза. Из минусов, на прозрачных полигонах (стекло, вода) данный метод не работает. Существуют улучшенные решения, убирающие данный минус от AMD – AAA, и от NVIDIA – TrAA.

Методы сглаживания пост-обработка:

- Следующий этап развития это **FXAA.** Этот алгоритм использует мощность потоковых процессоров при постфильтрации и производится совместно с другими фильтрами, вроде размытия в движении. При FXAA совершается один проход по всем пикселям изображения и усредняются цвета соседних пикселей. Метод FXAA менее требователен, чем MSAA: обеспечивает четкость изображения, сравнимую с 8xMSAA, при равных с 2xMSAA затратах производительности. В этом как раз основная польза метода FXAA – оно обеспечивает значительно лучшую скорость при примерно таком же качестве сглаживания, что и у MSAA.
- Существует аналог FXAA от Intel – **MLAA** *(MorphoLogical anti-aliasing)* это единственное сглаживание, работающее полностью на процессоре, усредняются цвета только пикселей на краях.

К гибридной мультивыборке относится **SMAA**, это сочетание FXAA и MLAA. По сути же несколько улучшенный MLAA, но работа-

ющий на видеокарте. Методы мульти-выборки и пост-обработки можно включить одновременно, поскольку один метод основан на рендеринге, а другой – на постобработке. Но SMAA делает более эффективное объединение двух методов при реализации временного фильтра.

Методы временной фильтрации:

- Чтобы улучшить качество сглаживания, почти не повысив ресурсоёмкость, в NVIDIA был разработан ещё один метод – **TXAA**. В отличие от других типов сглаживания, которые работают только с одним кадром (то есть с неподвижным изображением), это умеет работать с движущимися объектами. Основная цель TXAA в том, чтобы добиться качества сглаживания, максимально близкого к тому, что делается в пререндеренной графике (полнометражные мультфильмы и эффекты в кино). TXAA— гибридный метод, который включает как использование аппаратных MSAA мультисэмплов, так и специальный качественный сглаживающий постфильтр и даже опциональную временную компоненту. Метод сглаживания TXAA доступен в двух режимах: TXAA 1 и TXAA 2. Первый режим предлагает качество сглаживания, аналогичное методу 8x MSAA, но с производительностью, идентичной 2xMSAA, а второй обеспечивает ещё лучшее качество, но с производительностью, примерно соответствующей 4xMSAA.
- **MFAA** (Multi-Frame Anti-Aliasing) – технология многокадрового сглаживания. Не использует фильтры-шейдеры. Качество как у 4xMSAA по производительности соответствует 2xMSAA.

Так же существует еще множество менее известных улучшений перечисленных методов от разных компаний. Каждое из них, это поиск баланса между производительностью и качеством графики.

Так же необходимо упомянуть технологии, которые косвенно относятся к технологиями трехмерной графики, такие как использование нескольких видеокарт, синхронизация кадров, многомониторный вывод и трассировка лучей для профессиональных решений.

NVIDIA SLI (AMD CrossFireX) – технология, позволяющая использовать мощности нескольких видеокарт для обработки трёхмерного изображения. SLI-систему можно организовать двумя способами:

- С помощью специального мостика SLI;
- Программным путём.

Существуют следующие методы обработки изображений с помощью нескольких видеокарт:

- **Scissor.** Изображение разбивается на несколько частей, количество которых соответствует количеству видеокарт в связке
- **Alternate Frame Rendering.** Одна видеокарта обрабатывает только чётные кадры, а вторая – только нечётные
- **SuperAA.** Один и тот же кадр генерируется на всех видеокартах с разными шаблонами сглаживания.
- **SuperTiling.** Изображение разбивается на квадраты 32x32 пикселя и принимает вид шахматной доски. Каждый квадрат обрабатывается одной видеокартой.

Вертикальная синхронизация (VSync) придумана и используется для того, чтобы минимизировать артефакты изображения в виде разрывов кадра, заметные тогда, когда частота кадров вырастает выше частоты обновления монитора. Технологии синхронизации кадров NVIDIA G-SYNC и AMD FreeSync динамически синхронизируют скорость регенерации дисплея с частотой смены кадров GPU. NVIDIA Adaptive VSync – выключает вертикальную синхронизацию, когда частота смены кадров падает меньше частоты обновления монитора.

Современные видеокарты имеют расширенную поддержку **многомониторного вывода,** что позволяет подключать несколько мониторов к одной видеокарте.

Декодирование и последующая обработка видео на графическом процессоре (NVIDIA PureVideo, ATI Avivo). Раньше это делал центральный процессор.

Производители видеокарт заботятся об энергопотреблении, поэтому введено множество технологий **снижения энергопотребления** видеокарт: AMD Enduro, Radeon Chill, NVIDIA WHISPERMODE, NVIDIA BatteryBoost, NVIDIA Optimus.

NVIDIA OptiX (AMD Radeon Rays) представляет собой работающее на основе метода трассировки лучей программное обеспечение, ускоренное за счет использования ресурсов графического процессора. Технология имитирует эффект отражения и преломления лучей света в среде и их взаимодействия с виртуальными объектами, создавая фотореалистичные 3D-изображения.

6 МНОГОПРОЦЕССОРНЫЕ И МНОГОЯДЕРНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

6.1 Многоядерные структуры процессора и многопотоковая обработка команд

Корпорация Intel, лидер в разработке микропроцессоров с x86 архитектурой, ежегодно на протяжении долгого времени увеличивала производительность своих процессоров преимущественно за счет увеличения тактовой частоты и использования гиперконвейерной технологии выполнения команд, что, в свою очередь, значительно увеличивало энергопотребление и соответственно количество выделяемой процессором тепловой энергии. Это привело к тому, что компания уперлась в энергетический предел, ограничивающий возможности наращивания производительности процессорных кристаллов традиционными способами. Перед компанией Intel остро встала проблема разрешения противоречия между производительностью процессора и энергопотреблением.

Использование многоядерных структур процессора является одним из путей решения этой проблемы. Совмещение в одном процессоре двух вычислительных ядер позволяет удерживать рассеиваемую им мощность в допустимых пределах за счет сравнительно незначительного понижения тактовой частоты ядер: при снижении рабочей частоты на 20 % производительность ядра падает примерно на 13 %, а энергопотребление – на 50 %. При этом двухъядерный процессор все равно существенно выигрывает в производительности (при тех же условиях до 70 %) за счет увеличения количества команд, выполняемых в процессоре за один такт, но для этого необходимо на программном уровне обеспечить загрузку обоих ядер, для чего требуется соответствующая оптимизация программного кода.

Первыми стали использовать двухъядерные структуры разработчики RISC-процессоров:

- компания IBM (процессоры Power 4, 5, Power PC G5);
- Sun Microsystems (процессор Ultra Sparc IV).

В настоящее время выпускается достаточно большое количество типов многоядерных процессоров различных фирм производителей с 2, 4, 6, 8, 12 ядрами. Можно сказать, что в развитии вычислительной техники с 2005 г. наступила эра использования многоядерных структур процессоров.

Другим направлением развития микропроцессорной индустрии на ближайшие годы будет многопоточность. Двупотоковая обработка ко-

манд на одном процессоре (ядре) основывается на том, что в каждый момент времени только часть ресурсов процессора (ядра) используется при выполнении программного кода. Неиспользуемые ресурсы также можно загрузить работой, например задействовать для параллельного выполнения еще одного приложения. В этом случае операционная система (ОС) и приложения «видят» именно два логических процессора (ядра) и могут распределять работу между ними, как и в случае полноценной двухпроцессорной системы (рис. 6.1).

Для того чтобы использовать технологии многопоточности, необходимы эффективные компиляторы, которые разработаны и поставляются вместе с микропроцессорами.

Технологии многопоточности в настоящее время используются различными фирмами:

- Intel – технология Hyper-Threading (HT), технология Simultaneous multithreading (SMT);
- Sun/Oracle– технология Chip Multithreading (CMT);
- Fujitsu Siemens Computer – технология Vertical Multithreading (VMT).

Применение многоядерной структуры одновременно с технологией многопоточности увеличивает количество используемых логических процессоров (ядер) в 2 раза (Core i7, Itanium 2, Xeon), в 4 раза (Ultra SPARC T1), в 8 раз (Ultra SPARC T2), что существенно увеличивает производительность физического процессора.

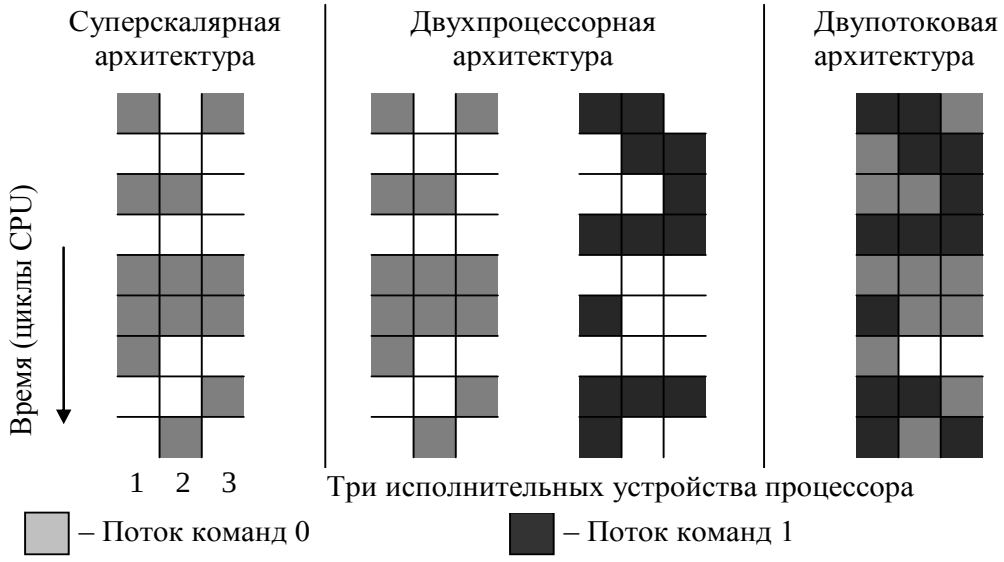


Рис. 6.1. Многопоточность в сравнении с другими способами обработки команд

Аппаратная многопоточность

Аппаратная многопоточность позволяет нескольким потокам совместно использовать функциональные блоки одного и того же процессора. Чтобы разрешить подобное совместное использование, процессор должен копировать независимое состояние каждого потока. Например, у каждого потока должна быть отдельная копия файла регистров и счетчика команд. Память как таковая должна совместно использоваться посредством механизмов виртуальной памяти, которые уже имеют мультипрограммную поддержку. Кроме этого оборудование должно поддерживать возможность достаточно быстрого переключения между разными потоками. В частности, переключение потока должно быть намного более эффективным, чем переключение процессов, для которого обычно требуются от сотен до тысяч процессорных циклов, а переключение потоков должно быть мгновенным.

Существуют два основных подхода к аппаратной многопоточности. **Мелкомодульная многопоточность** (Fine-grained multithreading) производит переключение между потоками на каждой инструкции, что приводит к чередующемуся выполнению нескольких потоков. Это чередование часто осуществляется по кругу с пропуском любых приостановленных потоков. Чтобы мелкомодульная многопоточность стала практичной, процессор должен быть в состоянии переключать потоки с каждым тактовым циклом. Одно из основных преимуществ мелкомодульной многопоточности заключается в том, что она может скрадывать потери пропускной способности, возникающие как из-за коротких, так и из-за длительных приостановок, поскольку пока один из потоков приостановлен, будут выполняться инструкции других потоков. Основной недостаток мелкомодульной многопоточности заключается в том, что она замедляет выполнение отдельно взятых потоков, поскольку поток, готовый к выполнению без остановок, будет замедлен инструкциями других потоков.

В качестве альтернативы мелкомодульной многопоточности была прдумана **крупномодульная многопоточность** (coarse-grained multithreading). При использовании крупномодульной многопоточности потоки переключаются только ири значительных приостановках, например при промахах при обращении к кэш-памяти второго уровня. Это изменение освобождает от необходимости не иметь никаких затрат на переключение потоков и существенно снижает вероятность замедления ныполнения отдельного потока, поскольку инструкции из других потоков могут быть запущены, только когда попадется весьма за гратная приостановка. Но крупномодульная многопоточность страдает от

другого существенного недостатка: она ограничена в возможностях преодоления потерь пропускной способности, особенно от коротких приостановок. Это ограничение является результатом стоимости конвейерного запуска крупномодульной многопоточности. Поскольку процессор с крупномодульной многопоточностью выдает инструкции из одного потока, то в случае приостановки конвейер должен быть очищен или заморожен. Новый поток, который начинает выполняться после приостановки, должен заполнить конвейер до того, как инструкции получат возможность завершаться. Из-за этих пусковых издержек крупномодульная многопоточность больше всего подходит для сокращения издержек дорогостоящих приостановок, когда перезаполнением конвейера на фоне времени приостановки можно пренебречь.

Параллельная многопоточность (simultaneous multithreading. SMT) является вариантом аппаратной многопоточности, которая использует ресурсы процессора с параллельным запуском инструкций и динамической диспетчеризацией с тем, чтобы воспользоваться параллелизмом науровне потоков и в тоже время воспользоваться параллелизмом на уровне инструкций. Основной замысел, ставший поводом для разработки SMT, заключается в том, что процессоры с параллельным запуском инструкций зачастую обладают более высокой степенью параллелизма функциональных блоков, чем та, которую может эффективно использовать один поток. Более того, используя переименование регистров и динамическую диспетчеризацию, несколько инструкций из разных потоков могут быть запущены без учета их взаимозависимости; разрешение зависимостей может быть произведено за счет возможностей динамической диспетчеризации.

В силу своей зависимости от существующих динамических механизмов, SMT не производит переключения ресурсов при каждом цикле. Вместо этого SMT всегда выполняет инструкции из нескольких потоков, поручая оборудованию связывать слоты инструкций и переименовывать регистры в соответствии с потоками, которым они принадлежат.

6.2 Архитектуры вычислительных систем

Точно также, как однопроцессорные компьютеры представлены по классификации М. Флина архитектурами с одним потоком данных SISD и множеством потоков данных SIMD, так и многопроцессорные системы могут быть представлены двумя базовыми типами архитектур в зависимости от параллелизма данных:

- **MISD (Multiple Instruction Single Data)** – множество потоков команд – один поток данных;
- **MIMD (Multiple Instruction Multiple Data) –** множество потоков команд – множество потоков данных.

Класс MISD долгое время пустовал, поскольку не существовало практических примеров реализации систем, в которых одни и те же данные обрабатываются большим числом параллельных процессов. В дальнейшем для MISD нашлась адекватная организация вычислительной системы – распределённая мультипроцессорная система с общими данными. Наиболее простая и самая распространённая система этого класса – обычная локальная сеть персональных компьютеров, работающая с единой базой данных, когда много процессоров обрабатывают один поток данных. Впрочем, тут есть одна тонкость. Как только в такой сети все пользователи переключаются на обработку собственных данных, недоступных для других абонентов сети, MISD-система превращается в систему с множеством потоков команд и множеством потоков данных, соответствующую MIMD-архитектуре.

Так как только MIMD-архитектура включает все уровни параллелизма – от конвейера операций до независимых заданий и программ, то любая вычислительная система этого класса в частных приложениях может выступать как SISD- и SIMD-система. Например, если многопроцессорный комплекс выполняет одну единственную программу без каких-либо признаков векторного параллелизма данных, то в этом конкретном случае он функционирует как обычный SISD-компьютер, и весь его потенциал остается невостребованным. Таким образом, употребляя термин «MIMD», надо иметь в виду не только много процессоров, но и множество вычислительных процессов, одновременно выполняемых в системе.

Другая классификация многопроцессорных вычислительных систем (МВС) основана на разделении МВС по двум критериям: **способу построения памяти** (общая или распределенная) и **способу передачи информации**. Основные типы машин представлены в табл. 6.1. Здесь приняты следующие обозначения: *P* – элементарный процессор, *M* – элемент памяти, *K* – коммутатор, *C* – кэш-память.

Параллельная вычислительная система с общей памятью и шинной организацией обмена (машина 1) позволяет каждому процессору системы «видеть», как решается задача в целом, а не только те части, над которыми он работает. Общая шина, связанная с памятью, вызывает серьёзные проблемы для обеспечения высокой пропускной способности каналов обмена. Одним из способов обойти эту ситуацию является ис-

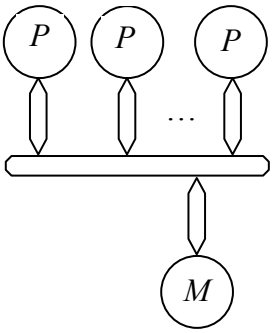
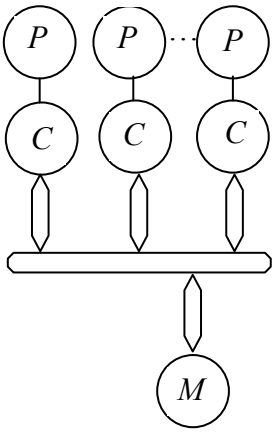
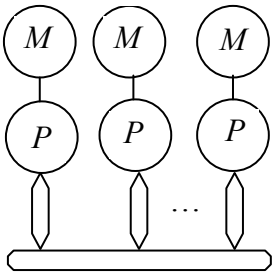
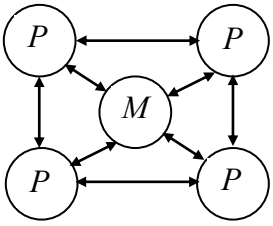
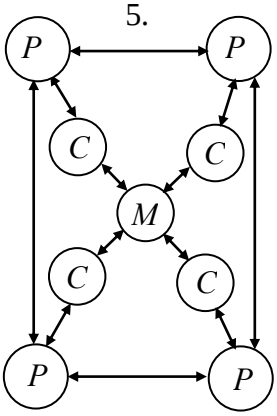
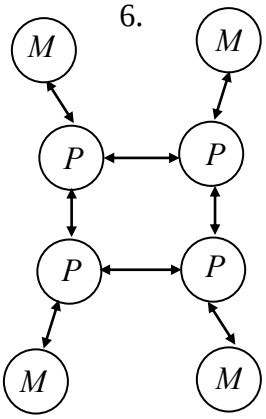
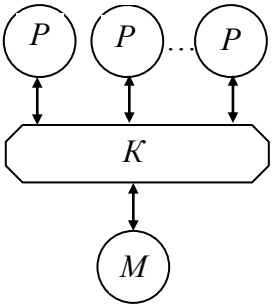
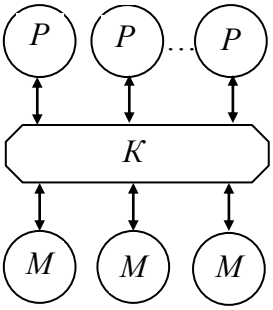
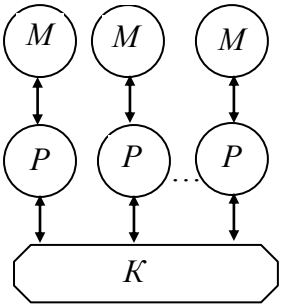
пользование кэш-памяти (машина 2). В этом случае возникает проблема когерентности (адекватности) содержимого кэш-памяти и основной памяти. Другим способом повышения производительности систем является отказ от общей памяти (машина 3).

Идеальной машиной является вычислительная система, у которой каждый процессор имеет прямые каналы связи с другими процессорами, но в этом случае требуется чрезвычайно большой объём оборудования для организации межпроцессорных обменов. Определенный компромисс представляет сеть с фиксированной топологией, в которой каждый процессор соединен с некоторым подмножеством процессоров системы (машины 4, 5, 6).

Если процессорам, не имеющим непосредственного канала обмена, необходимо взаимодействовать, они передают сообщения через промежуточные процессоры. Одно из преимуществ такого подхода – не ограничивается рост числа процессоров в системе. Недостаток – требуется оптимизация прикладных программ, чтобы обеспечить выполнение параллельных процессов, для которых необходимо активное воздействие на соседние процессоры.

Наиболее интересным вариантом для перспективных параллельных вычислительных комплексов является сочетание достоинства архитектур с распределенной памятью и каналами межпроцессорного обмена. Один из возможных методов построения таких комбинированных архитектур – конфигурация с коммутацией, когда процессор имеет локальную память, а соединяются процессоры между собой с помощью коммутатора (машина 9). Коммутатор может оказаться весьма полезным для группы процессоров с распределяемой памятью (машина 8). Данная конфигурация похожа на машину с общей памятью (машина 7), но здесь исключены проблемы пропускной способности шины.

Таблица 6.1. Основные типы машин

Типы передачи сообщений	Типы памяти		
	Общая память	Общая и распределенная	Распределенная память
Шинные соединения	1. 	2. 	3. 
Фиксированные перекрестные соединения	4. 	5. 	6. 
Коммутационные структуры	7. 	8. 	9. 

MIMD-системы по способу взаимодействия процессоров (рис. 6.2) делятся на **системы с сильной и слабой связью.**

Системы с сильной связью (иногда их называют «истинными» мультипроцессорами) основаны на объединении процессоров на общем поле оперативной памяти.

Системы со слабой связью представляются многопроцессорными и многомашинными системами с распределенной памятью. Разница организации MIMD-систем с сильной и слабой связью проявляется при обработке приложений, отличающихся интенсивностью обменов между процессами.

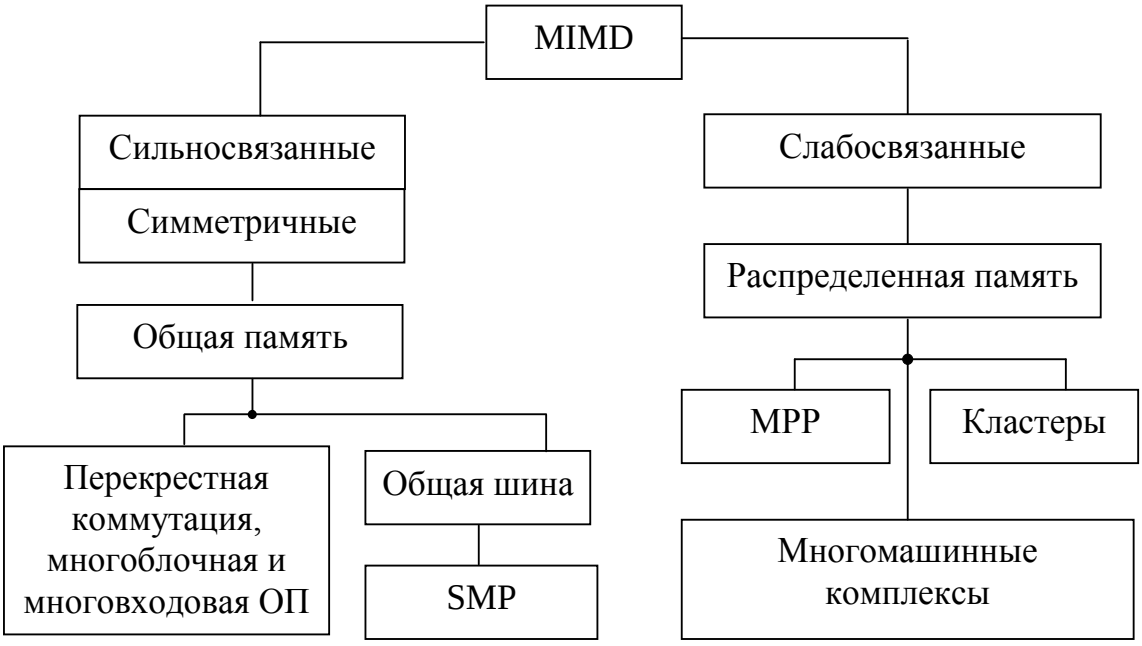


Рис. 6.2. Классификация вычислительных систем с MIMD-архитектурой

6.2.1 Сильносвязанные многопроцессорные системы

В архитектурах многопроцессорных сильносвязанных систем можно отметить две важнейшие характеристики: **симметричность** (равноправность) всех процессоров системы и **распределение** всеми процессорами **общего поля оперативной памяти.**

В таких системах, как правило, число процессоров невелико (не больше 16) и управляет ими централизованная операционная система. Процессоры обмениваются информацией через общую оперативную память. При этом возникают задержки из-за межпроцессорных конфликтов. При создании больших мультипроцессорных ЭВМ (мэйнфреймов, суперЭВМ) предпринимаются огромные усилия по увеличе-

нию пропускной способности оперативной памяти (перекрестная коммутация, многоблочная и многовходовая оперативная память и т.д.). В результате аппаратные затраты возрастают чуть ли не в квадратичной зависимости, а производительность системы упорно «не желает» увеличиваться пропорционально числу процессоров. То, что могут себе позволить дорогостоящие и сложные мэйнфреймы и суперкомпьютеры, не годится для компактных многопроцессорных серверов.

Микропроцессоры с единым адресным пространством бывают двух типов. Первый из них характеризуется одинаковый временем обращения к оперативной памяти, независимо от того, какой из процессоров запрашивает это обращение и какое именно слово запрашивается. Такие машины называются мультипроцессорами **с однородным доступом к памяти** (uniform memory access, UMA). У второго типа часть адресов памяти имеет намного более высокое быстродействие, в зависимости от того, какой процессор какое слово запросил. Такие машины называют мультилроцессорами **с неоднородным доступом к памяти** (nonuniform memory access, NUMA).

По мере того как работающие в параллельном режиме процессоры будут использовать общие данные, им понадобится координация работы с общими данными, в противном случае один процессор может приступить к работе с данными еще до того, как другой завершит работу с ними. Такая координация называется **синхронизацией.**

Архитектура SMP

Для простой и «дешевой» поддержки многопроцессорной организации была предложена **архитектура SMP –** мультипроцессирование с разделением памяти, предполагающая объединение процессоров на общей шине оперативной памяти. За аппаратную простоту реализации средств SMP приходится расплачиваться процессорным временем ожидания в очереди к шине оперативной памяти. В большинстве случаев пользователи готовы добавить в сервер один или более процессоров (но редко – более четырёх) в надежде увеличить производительность системы. Стоимость этой операции ничтожна по сравнению со стоимостью всего сервера, а результат чаще всего оправдывает ожидания пользователя.

Пропускную способность памяти в таких системах можно значительно увеличить путём применения больших многоуровневых кэшей. При этом кэши могут содержать как **разделяемые**, так и **частные данные**. Частные данные – это данные, которые используются одним процессором, в то время как разделяемые данные используются многими

процессорами, по существу обеспечивая обмен между ними. Если кэшируются разделяемые данные, то они реплицируются и могут содержаться в нескольких кэшах. Кроме сокращения задержки доступа и требуемой полосы пропускания, такая репликация данных способствует также общему сокращению количества обменов. Однако кэширование разделяемых данных вызывает новую проблему: **когерентность кэш-памяти.** Эта проблема возникает из-за того, что значение элемента данных в памяти, используемое двумя разными процессорами, доступно этим процессорам только через их индивидуальные кэши. На рис. 6.3 показан простой пример, иллюстрирующий эту проблему, где A' и B' – кэшированные копии элементов A и B в основной памяти. Когерентное (адекватное) состояние кэша и основной памяти, когда $A' = A$ & $B' = B$, изображено на рис. 6.3, *а*. Во втором случае (рис. 6.3, *б*) предполагается использование кэш-памяти с обратной записью, когда ЦП записывает значение 550 в ячейку A'. В результате A' содержит новое значение, а в основной памяти осталось старое значение – 100. При попытке вывода A из памяти будет получено старое значение.

В третьем случае (рис. 6.3, *в*) подсистема ввода/вывода вводит в ячейку памяти B новое значение 440, а в кэш-памяти осталось старое значение B.

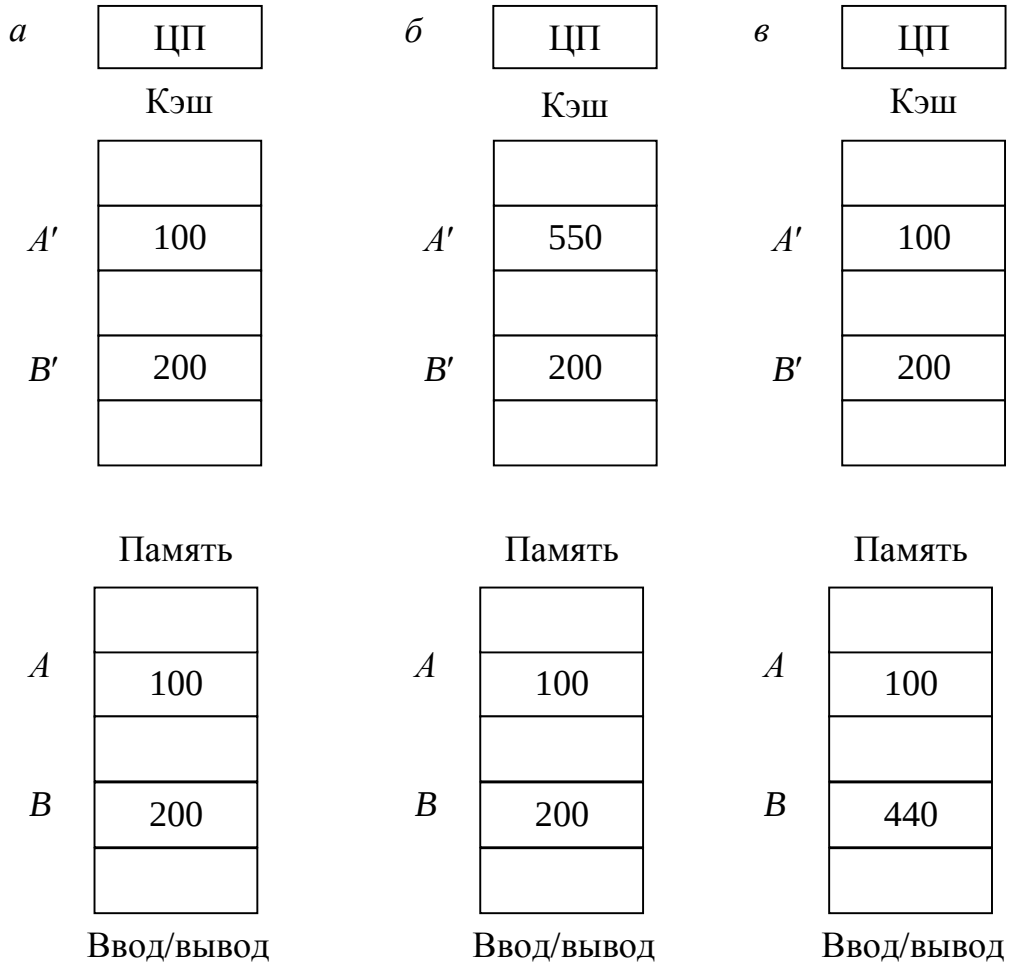


Рис. 6.3. Кэш и память когерентны: $A' = A$ & $B' = B$ (*а*), кэш и память некогерентны: $A' \neq A$ (*б*), кэш и память некогерентны: $B' \neq B$ (*в*)

Обычно в малых мультипроцессорах (с небольшим количеством процессоров) используется аппаратный механизм, называемый **протоколом когерентности кэш-памяти**, позволяющий решить эту проблему.

Основное преимущество SMP – **относительная простота программирования**. В ситуации, когда все процессоры имеют одинаково быстрый доступ к общей памяти, вопрос о том, какой из процессоров какие вычисления будет выполнять, не столь принципиален, и значительная часть вычислительных алгоритмов, разработанных для последовательных компьютеров, может быть ускорена с помощью распараллеливающих и векторизирующих трансляторов.

Архитектура SMP стала своего рода стандартом для всех современных многопроцессорных серверов.

6.2.2 Слабосвязанные многопроцессорные системы

Существует несколько способов построения крупномасштабных систем с распределённой памятью.

1. Многомашинные системы. В таких системах отдельные компьютеры объединяются либо с помощью сетевых средств, либо с помощью общей внешней памяти (обычно – дисковые накопители большой емкости).

2. Системы с массовым параллелизмом MPP (Massively Parallel Processor). Идея построения систем этого класса тривиальна: берутся серийные микропроцессоры, снабжаются каждый своей локальной памятью, соединяются посредством некоторой коммуникационной среды, например сетью.

Системы с массовым параллелизмом могут содержать десятки, сотни и тысячи процессоров, объединённых коммутационными сетями самой различной формы – от простейшей двумерной решетки до гиперкуба. Достоинства такой архитектуры: во-первых, она использует стандартные микропроцессоры; во-вторых, если требуется высокая терафлопсная производительность, то можно добавить в систему необходимое количество процессоров; в-третьих, если ограничены финансы или заранее известна требуемая вычислительная мощность, то легко подобрать оптимальную конфигурацию.

Однако есть и решающий «минус», сводящий многие «плюсы» на нет. Дело в том, что межпроцессорное взаимодействие в компьютерах этого класса идет намного медленнее, чем происходит локальная обработка данных самими процессорами. Именно поэтому написать эффективную программу для таких компьютеров очень сложно, а для некоторых алгоритмов иногда просто невозможно.

3. Кластерные системы. Данное направление, строго говоря, не является самостоятельным, а, скорее, представляет собой комбинацию из архитектур SMP и MPP. Из нескольких стандартных микропроцессоров и общей для них памяти формируется вычислительный узел (обычно по архитектуре SMP). Для достижения требуемой вычислительной мощности узлы объединяются высокоскоростными каналами.

Эффективность распараллеливания процессов во многих случаях сильно зависит от топологии соединения процессорных узлов. Идеальной является топология, в которой любой узел мог бы напрямую связаться с любым другим узлом. Однако в кластерных системах это технически трудно реализуемо. Обычно процессорные узлы в современных кластерных системах образуют или двумерную решетку, или гиперкуб.

Для синхронизации параллельно выполняющихся в узлах процессов необходим обмен сообщениями, которые должны доходить из любого узла системы в любой другой узел. При этом важной характеристикой является максимальное расстояние между узлами. Если сравнивать по этому параметру двумерную решетку и гиперкуб, то при увеличении числа узлов топология гиперкуба является более выгодной.

Время передачи информации от узла к узлу зависит от стартовой задержки и скорости передачи. Прогресс в производительности процессоров гораздо больше, чем в пропускной способности каналов связи. За время передачи процессорные узлы успевают выполнить большое количество команд. Поэтому инфраструктура каналов связи является одной из главных компонент кластерной или MPP-системы.

Благодаря маштабируемости именно кластерные системы являются сегодня лидерами по достигнутой производительности.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ ДЛЯ САМОПРОВЕРКИ

1. Дайте определение ЭВМ, вычислительной и информационной системам, архитектуре ЭВМ.
2. Опишите развитие и классификацию архитектур компьютеров.
3. Опишите конвейерную технологию выполнения команд.
4. Охарактеризуйте характерные черты суперскалярной обработки команд.
5. Приведите классификацию архитектуры SISD с характеристикой классов.
6. Определите основные характерные черты CISC-архитектуры.
7. Охарактеризуйте основные характерные черты RISC-архитектуры.
8. Укажите основные характерные черты VLIW-архитектуры.
9. Охарактеризуйте основные отличительные черты EPIC-концепции.
10. В чем суть матричного и векторно-конвейерного способов организации SIMD-архитектуры?
11. В чем суть MMX-технологии и потоковых SIMD-расширений?
12. Почему появились многоядерные структуры процессоров и технологии многопоточности?
13. Охарактеризуйте все виды производительности компьютера.
14. Как определить энергоэффективность процессора?
15. Приведите классификацию ЭВМ по назначению.
16. Приведите классификацию ЭВМ по функциональным возможностям.
17. Опишите функциональные возможности, области применения, современные разработки мэйнфреймов.
18. Опишите функциональные возможности, пути развития, современные разработки суперЭВМ.
19. Приведите классификацию микроЭВМ с краткой характеристикой классов.
20. Опишите функциональные возможности, назначение, платформы рабочих станций.
21. Приведите классификации серверов с пояснениями.
22. Перечислите требования, которые учитываются при проектировании серверов.

23. Охарактеризуйте класс модульных серверов.

24. Особенности многоузловых серверных систем.

25. Какими преимуществами обладают блейд-серверы?

26. Какими характеристиками должен обладать ПК?

27. Приведите классификацию ПК по способу использования и по назначению.

28. Дайте классификацию ноутбуков.

29. Охарактеризуйте планшетные ПК.

30. Приведите классификацию, состав, платформы карманных устройств.

31. Охарактеризуйте встраиваемые и промышленные компьютеры.

32. Опишите обобщенную структуру ЭВМ и пути ее развития.

33. Опишите типы данных в интеловских процессорах с IA-32, IA64.

34. Опишите типы данных в IA-64.

35. Опишите векторные типы данных.

36. Охарактеризуйте классические структуры команд.

37. Объясните суть ассоциативного и адресного поиска операндов в памяти.

38. Как осуществляется непосредственная, прямая и косвенная адресация операндов?

39. Каким образом реализуется адресация операндов «базирование способом суммирования» и «базирование способом совмещения»?

40. Как реализуется индексная адресация?

41. Как осуществляется адресация операндов «базирование с индексированием»?

42. Опишите развитие CISC-системы команд x86 (по годам).

43. Какие новые возможности появились у процессора с введением расширения команд SSE-2, SSE-3, SSE-4?

44. Опишите расширенеие AES-NI, AVX.

45. Опишите расширение AVX2, FMA3, BMI, MIC и SGX.

46. Охарактеризуйте особенности режимов работы процессоров AMD64, Intel64.

47. Опишите обобщенный формат команд x86 и форматы команд RISC-процессора.

48. Приведите формат команд IA-64 и структуру пакета инструкций.

49. Опишите принципы организации системы прерывания программ.

50. Приведите характеристики системы прерывания.

51. Как реализуется программно-управляемый приоритет прерывающих программ?

52. Определите назначение и структуру центрального процессора ЭВМ.

53. Приведите классификацию методов построения центрального устройства управления процессора.

54. Определите назначение, структуру, количество основных функциональных регистров IA-32 и регистров блока обработки чисел с плавающей точкой IA-32.

55. Определите назначение, структуру, количество регистров MMX-технологии и расширений SSE, SSE2.

56. Объясните суть процедуры переименования регистров в процессорах.

57. Опишите регистровые структуры процессоров AMD64, Intel64.

58. Опишите регистровые структуры процессоров IA-64.

59. Сформулируйте характерные черты современных универсальных микропроцессоров.

60. Опишите стратегию развития процессоров Intel.

61. Приведите особенности микроархитектуры процессоров Intel Skylake.

62. Опишите историю создания и современное состояние семества микропроцессоров Эльбрус.

63. Опишите иерархическую структуру памяти компьютера.

64. Охарактеризуйте способы размещения данных в кэш-памяти.

65. Какие существуют методы обновления строк в основной и кэш-памяти?

66. Какие существуют методы замещения строк в кэш-памяти?

67. Опишите общие принципы организации оперативной памяти компьютера.

68. Охарактеризуйте способы распределения оперативной памяти.

69. Какие существуют методы повышения пропускной способности ОП?

70. Сформулируйте концепцию виртуальной памяти.

71. Опишите страничное и странично-сегментное распределение виртуальной памяти.

72. Перечислите характеристики интерфейсов.

73. Приведите классификацию интерфейсов.

74. Охарактеризуйте способы передачи данных в подсистеме ввода-вывода.

75. Опишите системную организацию на базе чипсетов компании Intel.

76. Приведите классификацию MIMD-систем по способу взаимодействия процессоров.

77. Охарактеризуйте сильносвязанные и слабосвязанные многопроцессорные системы.

СПИСОК ЛИТЕРАТУРЫ

1. Чередов А.Д., Мальчуков А.Н. Организация ЭВМ и систем: учебное пособие / А.Д. Чередов; А.Н. Мальчуков Томский политехнический университет. – 4-е изд., перераб. и доп. – Томск: Изд-во Томского политехнического университета, 2016. – 236 с.

2. Паттерсон Д., Хеннесси Дж. Архитектура компьютера и проектирование компьютерных систем. / Д. Паттерсон,Дж. Хеннесси Классика Computer Science. 4-е изд. Спб.: Питер, 2012. 784 с.

3. Орлов С.А. Организация ЭВМ и систем: учебник для вузов / С.А. Орлов, Б.Я. Цилькер. – 3-е изд. – СПб.: Питер, 2014. – 688 с.

4. Новожилов О.П. Архитектура ЭВМ и системы. Учебное пособие для бакалавров. – М.: Изд-во Юрайт, 2015. – 527 с.

5. Бройдо В.Л. Вычислительные системы, сети и телекоммуникации: учебное пособие для вузов / В.Л. Брайдо, О.П. Ильина. – 4-е изд. – СПб.: Питер, 2011. – 555 с.

Интернет-ресурсы

6. Официальный сайт компании Intel, США. – http:// www.intel.com

7. Официальный сайт компании AMD, США. – http:// www.amd.com

8. Официальный сайт компании МЦСТ, Россия. – http://mcst.ru

9. Официальный сайт компании Nvidia, США. http:// www.nvidia.com

10. Официальный сайт компании HP, США. – http:// www.hp.com

11. Официальный сайт компании IBM, США. – http:// www.ibm.com

12. Сайт информационных технологий. – http:// www.ixbt.com

13. Сайт высоких технологий IT-индустрии. – http://citforum.ru

14. Сайт рейтинга ТОП-500. https://www.top500.org/

Учебное издание

МЫЦКО Евгений Алексеевич
АНДРЕЕВ Семён Алексеевич
ЧЕРЕДОВ Андрей Дмитриевич

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ

Учебное пособие

Научный редактор
доктор технических наук, профессор Н.Г. Марков

Редактор *Н.Т. Синельникова*

Верстка *Л.А. Егорова*

Отпечатано в Издательстве ТПУ в полном соответствии с качеством предоставленного оригинал-макета

Подписано к печати Формат 60×84/16.
Бумага «Снегурочка». Печать Xerox.
Усл. печ.л. 11,63. Уч.-изд. л. 10,53.
Заказ . Тираж 100 экз.

Национальный исследовательский
Томский политехнический университет
Система менеджмента качества
Издательства Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту BS EN ISO 9001:2008

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30.
Тел./факс: 8(3822)56-35-35, www.tpu.ru