В. А. Тихонов, А. В. Баранов

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ

Допущено Учебно-методическим объединением вузов РФ по университетскому политехническому образованию в качестве учебника для студентов высших учебных заведений, обучающихся по направлению 230100 «Информатика и вычислительная техника», специальности 230101 «Вычислительные машины, системы, комплексы и сети»

Москва
«Гелиос АРВ»
2008

УДК 682.3.06+519.68
ББК 32.973
О89

Рецензенты:
А. В. Забродин, чл.-кор. РАН, заместитель директора
ИПМ им. М.В. Келдыша РАН;
В. В. Корнеев, д-р техн. наук, проф., акад. Академии криптографии,
зам. начальника ФГУП «НИИ Квант»;
Кафедра «Вычислительная техника» МГИРЭА (ТУ)

Тихонов В.А., Баранов А.В.
О89 Организация ЭВМ и систем: Учебник / Под ред. акад. *В. К. Левина*. — М.: Гелиос АРВ, 2008. — 400 с., ил.

ISBN 978-5-85438-179-6

Учебник посвящен систематическому изложению принципов построения, архитектуре и компонентам современных ЭВМ и вычислительных систем. Соответствует программе одноименного курса.

Для студентов, обучающихся по специальности «Вычислительные машины, системы, комплексы и сети» (230101), может быть полезен и студентам других специальностей, где имеются соответствующие дисциплины, а также специалистам в области ЭВМ и вычислительных систем.

ББК 32.973

ISBN 978-5-85438-179-6

© Тихонов В. А., Баранов А. В., 2008
© Оформление. Издательство «Гелиос АРВ», 2008

ВВЕДЕНИЕ

Жизнь человека в настоящее время не представляется без использования ЭВМ. И их число и возможности растут из года в год. Современные информационные технологии (ИТ) на базе ЭВМ позволяют осуществлять доступ к необходимой информации практически из любой точки планеты.

Естественно, для того чтобы грамотно владеть всем арсеналом средств, созданных и создаваемых на базе ЭВМ, надо знать принципы построения ЭВМ и систем на их базе и их возможности.

Предлагаемый учебник «Организация ЭВМ и систем» позволит, по мнению авторов, более квалифицированно подойти к вопросам как построения ЭВМ и систем на их базе, так и к вопросам их эксплуатации и применения. Учебник написан в соответствии с программой дисциплины «Организация ЭВМ и систем», рассчитанной на 154 часа.

Исходя из требований Госстандарта в этой дисциплине должны быть отражены следующие вопросы:

- основные характеристики, области применения ЭВМ различных классов;
- функциональная и структурная организация процессора;
- организация памяти ЭВМ;
- основные стадии выполнения команды;
- конвейеризация;
- организация прерываний в ЭВМ;
- организация ввода-вывода;
- взаимодействие с периферийными устройствами;
- параллельные системы;
- понятие о многомашинных и многопроцессорных вычислительных системах.

Из этого перечня следует, что осветить все вопросы за отведенное время весьма проблематично. Поэтому некоторые из них будут рассмотрены фрагментарно. Это обусловлено тем, что дисциплина читается на старших курсах. К этому времени студенты уже изучили информатику, схемотехнику, микропроцессоры и ряд других дисциплин, где часть вопросов, указанных в стандарте, уже подробно рассмотрены.

В настоящее время выпущено достаточное количество научной и учебной литературы, посвященной рассматриваемой тематике.

К числу подобных изданий можно отнести:

1. *Корнеев В. В.* Вычислительные системы. — М.: Гелиос АРВ, 2004;

2. *Корнеев В. В.* Параллельные вычислительные системы. — М.: Нолидж, 1999;

3. *Орлов С. А., Цилькер Б. Я.* Организация ЭВМ и систем: Учебник. — СПб.: Питер, 2004.

Работы В. В. Корнеева являются монографиями, содержат большой фактический материал, но к учебникам предъявляются другие требования как по структуре, так и по содержанию.

Среди учебников по данному курсу следует отметить работу С. А. Орлова и Б. Я. Цилькера, имеющую гриф учебника Минобразования и науки. Этот учебник посвящен систематическому изложению вопросов организации структуры и функционирования ЭВМ и систем. Однако в нем большее внимание уделяется вопросам функционирования традиционных решений, хотя рассматриваются и перспективные в области организации и архитектуры ЭВМ и ВС.

По нашему мнению, среди недостатков этого учебника следует отметить следующие: нечеткость определения основных терминов. В первую очередь это относится к таким понятиям, как «организация», «структура» и другие, хотя на этот счет имеются руководящие документы различных уровней (государственные и международные); материал некоторых глав избыточен. Отдельные вопросы слушателям уже известны из других курсов; по структуре работа не полностью соответствует грифу «учебник»: мало примеров, некоторые вопросы изложены так, что их трудно понять без использования дополнительных источников; изложение некоторых вопросов напоминает отчеты о НИР.

Тем не менее этот учебник будет весьма полезным при изучении дисциплины как источник большого фактического материала.

Предлагаемый учебник состоит из 6 глав и введения.

В **первой главе** «Базовые сведения теории вычислительных машин, комплексов и систем» освещены вопросы, связанные с терминологией, дана классификация ЭВМ и ВС. Подробно рассмотрены показатели качества функционирования ВС.

Вторая глава посвящена ЭВМ последовательного типа. Здесь изложены вопросы, связанные со структурой и основными компонентами последовательной ЭВМ. Подробно описаны система команд ЭВМ, используемые способы адресации и система прерываний программ.

Третья глава посвящена организации памяти в ЭВМ и ВС.

В **четвертой главе** достаточно подробно рассмотрены вопросы, связанные с организацией ввода-вывода: шинная организация ввода-вывода; основные стандарты шин и их основные характеристики; магнитные и магнитно-оптические диски; дисковые массивы и уровни RAID; наиболее распространенные периферийные устройства.

Пятая глава, являющаяся наиболее объемной, посвящена параллельным ЭВМ и ВС. Здесь рассмотрены: типы параллельной обработки информации; используемые в настоящее время классификации параллельных ЭВМ и ВС; история развития параллелизма в архитектуре ЭВМ. Даны общие сведения о параллельном программировании. Подробно рассмотрены ВС классов SIMD, MIMD. Значительное внимание уделено вопросам конвейерной и векторной обработки дан-

ных, матричным, ассоциативным, кластерным ВС. Освещен вопрос, связанный с управлением вычислительными ресурсами в параллельных ЭВМ и ВС.

Шестая глава посвящена перспективным направлениям в развитии ЭВМ и ВС. Описано состояние разработок в области нейросистем, молекулярных, оптических и квантовых ЭВМ и ВС.

Материал учебника готовился по программе курса «Организация ЭВМ и систем» для специальности «Вычислительные машины, комплексы, системы и сети» (230101), он полностью может использоваться и для дисциплины «Вычислительные устройства и системы» специальности 201600.

Авторы выражают благодарность коллективу кафедры «Вычислительные системы специального назначения» ИКСИ Академии ФСБ России за участие и помощь в подготовке данной работы, а также Л. В. Барановой за помощь в подготовке иллюстраций.

ГЛАВА 1. БАЗОВЫЕ СВЕДЕНИЯ ТЕОРИИ ВЫЧИСЛИТЕЛЬНЫХ МАШИН, КОМПЛЕКСОВ И СИСТЕМ

1.1. ТЕРМИНОЛОГИЯ В ОБЛАСТИ ЭВМ, ВС И КОМПЛЕКСОВ

За более чем полувековую историю развития средства вычислительной техники (СВТ) прошли большой путь от простейших ЭВМ, производивших только целочисленные вычисления, до современных мощных вычислительных систем, способных обрабатывать практически любые виды информации. Не вдаваясь в историю развития СВТ (об этом написано достаточно много), отметим, что за ее время сменилось уже несколько поколений ЭВМ.

Хотя срок, прошедший со времени создания первой ЭВМ, и достаточно большой, однако вопросы терминологии в этой области пока что не нашли своего полного и окончательного решения. Поэтому авторы решили обобщить эти вопросы, опираясь на существующую нормативно-правовую базу.

Вопросы, связанные с терминологией, являются особенно важными при изучении новых дисциплин. Как заметил Г. Шухардт[1]: «Неясность терминологии оказывает такое же влияние на исследование, какое туман оказывает на мореплавание. Но она даже опаснее, потому что люди обычно не подозревают о ее существовании». Рассмотрим наиболее распространенные определения основных терминов в области ЭВМ и вычислительных систем (ВС).

Вычислительная машина (ВМ, Computer) — совокупность технических средств, создающая возможность проведения обработки информации и получения результата в необходимой форме. В состав ВМ входит и системное программное обеспечение (ПО) (ГОСТ 15971 90).

Электронно-вычислительная машина (ЭВМ, Electronic Computer) — программируемое функциональное устройство, состоящее из одного или нескольких взаимосвязанных центральных процессоров, периферийных устройств, управление которыми осуществляется посредством программ, располагающихся в оперативной памяти. Эта машина может производить большой объем вычислений, содержащих большое количество арифметических, логических и других операций без вмешательства пользователя в течение периода выполнения (стандарт ISO 2382/1-84).

В то же время ГОСТ 15971-90 определяет ЭВМ как вычислительную машину, основные функциональные устройства которой выполнены на электронных компонентах.

Система обработки данных (СОД) — совокупность технических средств и программного обеспечения, предназначенная для информационного обслужива-

[1] Г. Шухардт — видный австрийский лингвист и филолог (1842–1928). См. *Шухардт Г.* Избранные статьи по языкознанию / Пер. с нем. — М.: URSS, 2003.

ния пользователей и технических объектов. В состав технических средств входят сами ЭВМ, устройства сопряжения ЭВМ с объектами, аппаратура передачи данных и линии связи. Иногда вместо термина СОД используют понятие АСОИ (Автоматизированная система обработки информации, Computer-based information processing system).

По ГОСТ 15971-90 **система обработки информации (СОИ, Information Processing System)** — совокупность технических средств и ПО, а также методов обработки информации и действий персонала, обеспечивающая выполнение автоматизированной обработки информации.

По стандарту ISO 2382/1-84 СОД-система включает вычислительное оборудование и необходимый обслуживающий персонал и выполняет функции ввода-вывода, обработки, запоминания и управления для осуществления последовательности операций над данными.

СОД, настроенную на решение задач конкретной области применения, называют *вычислительной системой (ВС)*.

Вычислительная система (Computer System) — часть, подсистема автоматизированной системы обработки информации (АСОИ), конструктивно обособленная и автономная по своему функциональному назначению. Иногда под ВС понимается совокупность взаимосвязанных и согласованно действующих однородных или неоднородных ЭВМ и других устройств, обеспечивающих автоматизацию процессов приема исходной информации от ее источников, обработки информации и выдачи результатов обработки потребителю информации.

В соответствии со стандартом ISO 2382/1-84 «вычислительная система в общем случае включает по меньшей мере одну ЭВМ и соответствующее программное обеспечение и использует общую память для того, чтобы поместить в ней, полностью или частично программу, а также данные, необходимые для выполнения программы, введенные или запрошенные пользователем; выполняет манипуляции с данными, запрошенными пользователями, выполняет арифметические или логические операции; может выполнять самоизменяющиеся программы».

Вычислительный комплекс (ВК) — совокупность вычислительных и специальных средств, предназначенных для решения одной или нескольких широкомасштабных задач.

Многопроцессорная вычислительная система (Multiprocessor computer system) — система, в состав которой входят два или несколько процессоров.

Из текста приведенных определений видно, что они не являются строгими. Отсутствие строгости определений нельзя назвать недостатком, так как вычислительная техника — бурно развивающаяся отрасль, и строгое, четкое определение какого-либо устройства рискует быстро устареть вместе с определяемым устройством. Отметим, тем не менее, несколько размытую грань между понятиями «ЭВМ» и «вычислительная система». В самом деле, по определению стандарта ISO 2382/1-84, в состав ЭВМ может входить несколько центральных процессоров,

что согласно последнему данному нами определению позволяет рассматривать такую ЭВМ уже как многопроцессорную вычислительную систему. Впрочем, опять-таки согласно приведенным определениям любая ЭВМ представляет собой «вырожденный случай» вычислительной системы. В дальнейшем по тексту учебника в случае необходимости употребление терминов ЭВМ или ВС будет особо оговариваться.

Рассмотрим определения, касающиеся понятия **«архитектура»**. Термин «архитектура системы» часто употребляется как в узком, так и в широком смысле этого слова. В узком смысле под архитектурой понимается **архитектура набора команд**. Архитектура набора команд служит границей между аппаратурой и программным обеспечением (ПО) и представляет ту часть системы, которая видна программисту или разработчику компиляторов. Следует отметить, что это наиболее частое употребление термина. В широком смысле архитектура охватывает понятие организации системы, включающее такие высокоуровневые аспекты разработки компьютера, как систему памяти, структуру системной шины, организацию ввода-вывода и т. п.

По ГОСТ 15971-90 под **архитектурой вычислительной машины (ВМ)** понимается концептуальная структура ВМ, определяющая проведение обработки информации и включающая методы преобразования информации в данные и принципы взаимодействия технических средств и программного обеспечения.

Применительно к вычислительным системам термин **«архитектура»** может быть определен как распределение функций, реализуемых системой, между ее уровнями, точнее — как определение границ между этими уровнями. Таким образом, архитектура вычислительной системы предполагает многоуровневую организацию. В частности, архитектура первого уровня определяет, какие функции по обработке данных выполняются системой в целом, а какие возлагаются на внешний мир (пользователей, операторов, администраторов баз данных и т. д.). Система взаимодействует с внешним миром через **набор интерфейсов**: языки (язык оператора, языки программирования, языки описания и манипулирования базой данных, язык управления заданиями) и системные программы (программы-утилиты, программы редактирования, сортировки, сохранения и восстановления информации).

Исходя из вышеприведенных определений термина «архитектура» можно предложить следующее обобщающее определение.

Архитектура ЭВМ — абстрактное представление ЭВМ, которое отражает ее структурную, схемотехническую и логическую организацию. Понятие «архитектура ЭВМ» является комплексным и включает в себя целый ряд элементов, основные из них следующие:

1) структурная схема ЭВМ;

2) средства и способы доступа к элементам структурной схемы, включая обмен с внешней средой;

3) организация и разрядность интерфейсов в ЭВМ;
4) набор и доступность регистров;
5) организация и способы адресации памяти;
6) способы представления и форматы данных ЭВМ;
7) набор машинных команд;
8) форматы машинных команд;
9) обработка нештатных ситуаций (прерывания, особые ситуации, ловушки и т. д.);
10) топология связи отдельных устройств и модулей.

Таким образом, при разработке архитектуры ЭВМ условно можно выделить вопросы:

— общей структуры, организации вычислительного процесса и общения с машиной;

— логической организации представления, хранения и преобразования информации;

— логической организации совместной работы различных устройств;

— связанные с аппаратными и программными средствами машин.

К настоящему времени среди ЭВМ последовательного типа наибольшее распространение получили два типа архитектур. **Принстонская архитектура** разработана в Принстонском университете и предполагает единую память для хранения команд и данных. **Гарвардская архитектура** разработана соответственно в Гарвардском университете и характеризуется отдельной памятью для команд и отдельной — для данных.

Вычислительная сеть (сеть ЭВМ) (Computer Network) — территориально рассредоточенная многомашинная система, состоящая из взаимодействующих ЭВМ, связанных между собой каналами передачи данных.

Интерфейс (Interface) — совокупность средств и правил, обеспечивающих взаимодействие устройств ЭВМ или ВС, программ, а также пользователей (ГОСТ 15971-90).

Интерфейсы могут разграничивать определенные уровни внутри программного обеспечения. Например, уровень управления логическими ресурсами может включать реализацию таких функций, как управление базой данных, файлами, виртуальной памятью, сетевой телеобработкой. К уровню управления физическими ресурсами относятся функции управления внешней и оперативной памятью, управления процессами, выполняющимися в системе.

Следующий уровень отражает основную линию разграничения системы, а именно границу между системным программным обеспечением и аппаратурой. Эту идею можно развить и дальше и говорить о распределении функций между отдельными частями физической системы. Например, некоторый интерфейс определяет, какие функции реализуют центральные процессоры, а какие — контроллеры системных шин и магистралей передачи данных. Интерфейс следующе-

го уровня определяет разграничение функций между контроллерами системных шин и контроллерами внешних устройств. В свою очередь, можно разграничить функции, реализуемые контроллерами и самими устройствами ввода-вывода (терминалами, модемами, накопителями на магнитных и оптических дисках, сетевыми адаптерами). Архитектура таких уровней часто называется архитектурой физического ввода-вывода.

Исходя из вышеизложенного, можно считать, что **архитектура системы (Architecture of system)** — это ее описание на некотором общем уровне, включающее описание пользовательских возможностей программирования, системы команд, структуры системы, способов доступа к ней и пользовательского интерфейса, организации памяти и системы адресации, операций ввода-вывода, управления и т. д. Применительно к ВС термин «архитектура» используется для описания структуры, принципа действия, конфигурации и взаимного соединения основных элементов системы.

В соответствии с ГОСТ 15971-84 под **архитектурой ВС (Computing Architecture)** понимается логическая организация цифровой вычислительной системы, определяющая процесс обработки данных в конкретной ВС и включающая методы кодирования данных, состав, назначение, принципы взаимодействия технических средств и программного обеспечения. В повседневной практике под архитектурой ВС понимают ее состав и схему функциональных и управляющих связей между ее элементами.

Еще раз напоминаем, что архитектуру ЭВМ следует отличать от ее структуры, которая определяет конкретный состав ЭВМ на некотором уровне детализации.

Конфигурация СОИ — совокупность функциональных частей СОИ и связей между ними, обусловленная основными техническими характеристиками этих функциональных частей, а также требованиями решаемых задач.

1.2. КЛАССИФИКАЦИЯ ЭВМ И ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Классификация ЭВМ и ВС является довольно сложным процессом, так как существует слишком много признаков, по которым их можно классифицировать. При этом в разных ситуациях эти признаки имеют различное значение. Ранее ЭВМ классифицировались по стоимости, производительности, объему памяти и другим характеристикам, однако сегодня чаще всего используется классификация ЭВМ по областям применения. Для вычислительных систем — объектов в общем случае более сложных, чем ЭВМ, — число используемых классификационных признаков существенно больше.

1.2.1. Классификация ЭВМ

При рассмотрении классификации ЭВМ примем в качестве основного признака классификации область их применения. В соответствии с этим признаком можно выделить следующие классы ЭВМ:

- настольные ЭВМ (персональные ЭВМ и рабочие станции);

- портативные (переносимые) ЭВМ;
- серверы;
- мейнфреймы;
- супер-ЭВМ.

Следует отметить, что, помимо областей применения, данное разделение основывается также на существенном различии архитектур указанных классов ЭВМ.

1.2.1.1. Настольные ЭВМ (персональные ЭВМ и рабочие станции)

Под **персональным компьютером** или **персональной ЭВМ (ПК,** или **ПЭВМ**) понимают настольную ЭВМ, имеющую *эксплуатационные характеристики бытового прибора и универсальные функциональные возможности* (ГОСТ 15971-90).

Персональные компьютеры появились практически одновременно с началом массового выпуска микропроцессоров. Благодаря низкой стоимости ПК очень быстро завоевали твердые позиции на компьютерном рынке и создали предпосылки для разработки новых программных средств, ориентированных на конечного пользователя. Необходимо заметить, что отличительной особенностью ПК с самого начала их производства является *ориентация на широкого потребителя-непрофессионала*.

Под **рабочими станциями** обычно понимают настольные системы высокой производительности. Прародителями данного направления являются мини-ЭВМ, такие, как PDP-11 или VAX фирмы DEC. Основным отличием рабочих станций от ПК является *ориентация на профессиональных пользователей*. Рабочие станции — это хорошо сбалансированные системы, в которых высокое быстродействие сочетается с большим объемом оперативной и внешней памяти, высокопроизводительными внутренними магистралями, высококачественной и быстродействующей графической подсистемой и разнообразными устройствами ввода-вывода.

Несмотря на непрерывное интенсивное развитие вычислительной техники, на протяжении нескольких лет остается справедливой классификация настольных систем по ценовому признаку. По этому критерию различают:

- системы «начального уровня», которые строятся с использованием комплектующих, представленных на рынке в течение не менее 1–2 лет; моральное устаревание подобных ЭВМ происходит в течение 1–2 лет, но при этом данные модели имеют весьма низкую цену;
- системы «среднего уровня» строятся на базе наиболее распространенных на данный момент комплектующих, срок пребывания на рынке которых не превышает полугода; моральное устаревание данных систем происходит в течение 2–3 лет, тем не менее данный класс предлагает оптимальное соотношение «производительность/стоимость»;
- системы «высшего уровня» (high-end) строятся на базе наиболее производительных (и наиболее дорогих) на данный момент комплектующих; моральное

устаревание подобных систем происходит не раньше, чем через 4–5 лет, но, как правило, данные модели отличаются наиболее высокой ценой.

Как правило, с течением времени модели постепенно переходят из класса high-end в средний класс, и далее — в класс начального уровня. При этом, однако, производители комплектующих (в первую очередь центральных процессоров), стараются сразу позиционировать новые разработки для того или иного класса. Для систем начального уровня используются упрощенные архитектуры или не прошедшие испытания на высокие тактовые частоты изделия. Примером может служить линейка микропроцессоров Intel-Celeron («начальный уровень»), Pentium («средний класс») и Xeon («high-end»).

На сегодняшний день быстрый рост производительности ПК на базе новейших микропроцессоров в сочетании с постоянным снижением цен на эти изделия и развитием технологии локальных шин, позволяющей устранить многие «узкие места» в архитектуре ПК, делают современные персональные компьютеры весьма привлекательной альтернативой рабочим станциям. В свою очередь, рабочие станции «начального уровня» по стоимостным характеристикам близки к высокопроизводительным ПК.

Более того, в 1999 г. был введен в действие международный классификационный стандарт («спецификация 99»), фактически объединяющий ПК и рабочие станции. В соответствии с этим стандартом принято следующее деление ПК.

1. Массовый (Consumer PC).
2. Деловой (Office PC).
3. Портативный (Mobile PC).
4. Рабочая станция (Workstation PC).
5. Развлекательный (Entertainment PC).

Хотя грань между ПК и рабочими станциями постепенно стирается, отличительной особенностью последних остается внушительный набор дорогих периферийных устройств, состав которых зависит от области использования рабочей станции. Например, для профессиональной графической станции это будет, помимо мощного графического адаптера, монитор с большой диагональю и множеством поддерживаемых разрешений, планшетный сканер формата А3, цветной лазерный принтер. Кроме этого, рабочие станции от других настольных систем отличают повышенные надежность и отказоустойчивость, основанные на жестоком отборе высококачественных дорогих комплектующих и тщательнейшем тестировании готовых систем.

Классическим примером мощных графических рабочих станций являются ЭВМ фирмы Silicon Graphics Incorporated (SGI). По своим графическим характеристикам (скорость и качество обработки и отображения графической информации) изделия указанной фирмы на 3–5 лет опережают массовые разработки. Примером высоконадежных производительных рабочих станций могут служить ЭВМ фирмы Sun Microsystems.

1.2.1.2. Портативные ЭВМ

Как следует из названия, **портативные ЭВМ** отличаются малыми размерами и массой и, следовательно, возможностью переноса и работы в дороге. Поэтому данный класс ЭВМ еще называют **переносимыми ЭВМ (Mobile PC)**. В англоязычных источниках часто используется слово *laptop* — наколенный компьютер.

Портативные ЭВМ, хотя и являются персональными компьютерами, тем не менее выделены нами в отдельный класс. Связано это с существенными отличиями в архитектуре между ПК и портативными ЭВМ. Подробно эти отличия будут рассмотрены ниже, здесь же заметим, что, в свою очередь, портативные ЭВМ можно разделить на два больших подкласса — **карманные ПК** и **ноутбуки**.

История портативных ЭВМ ведет свое начало с малогабаритных ПК. Первым образцом считается модель 5100, изготовленная компанией IBM в 1975 г. Данная ЭВМ представляла собой полнофункциональный ПК, собранный в одном корпусе с клавиатурой и 5-дюймовым монохромным ЭЛТ-монитором. По тем временам это был настоящий переносимый микрокомпьютер, несмотря на внушительный вес 20 кг.

Отличительной особенностью **ноутбуков** (от английского *notebook* — «записная книжка») является изготовление их в виде раскладывающейся книги, на одной стороне которой располагается клавиатура, а на другой — экран монитора. **Ноутбук**— это *полностью автономная ЭВМ*, оборудованная встроенным дисплеем, клавиатурой, указательным устройством и аккумуляторной батареей.

Как отдельный массовый класс, ноутбуки выделились сравнительно недавно — с начала 90-х годов XX в. (точнее, в 1989 г. сразу несколько компаний представили новые модели портативных ПК, практически не отличающихся внешне от современных ноутбуков: NEC UltraLite, Zenith MinisPort и Compaq LTE). С этих пор рыночная доля ноутбуков непрерывно растет. При этом современный рынок ноутбуков характеризуется столь богатым многообразием, что назрела необходимость в отдельной классификации ноутбуков (рис. 1.1).

Ноутбуки можно классифицировать по следующим признакам:

1) по габаритам и массе;
2) по сегменту рынка;
3) по производительности и функциональным возможностям;
4) по типу центрального процессора и платформе.

Для портативного ПК **габариты** и **масса** являются одними из самым важных характеристик. По данному признаку ноутбуки можно разделить на:

- среднеформатные;
- «тонкие и легкие»;
- ультракомпактные;
- «замена настольного ПК».

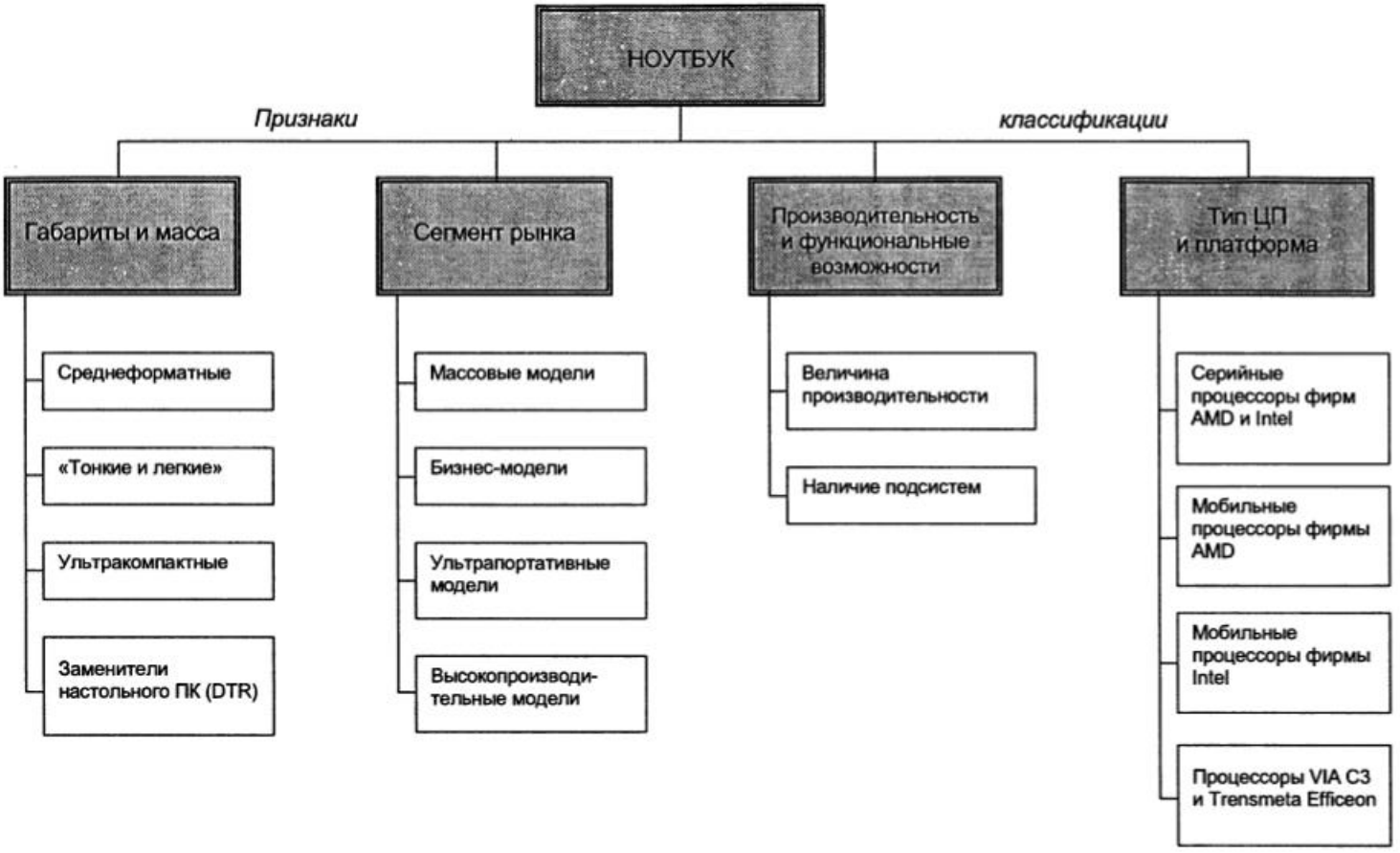


Рис. 1.1. Классификация ноутбуков

Большинство ноутбуков относится к *среднеформатным*, имеющим сравнительно небольшие габариты и массу 3–3,5 кг. Они представляют собой разумный компромисс между мобильностью и функциональными возможностями.

«*Тонкие и легкие*» ноутбуки предназначены для пользователей, совершающих частые разъезды и работающих в пути. Отличительными особенностями являются уменьшенные габариты и масса, позволяющие носить ноутбук вместе с документами. Отрицательными сторонами данных ПК являются уменьшенный шаг клавиатуры, минимум портов ввода-вывода и, как правило, отсутствие встроенных приводов накопителей на оптических и гибких магнитных дисках.

Ультракомпактные модели по размеру и весу сравнимы с небольшой книжкой. Платой за миниатюрность является, помимо недостатков тонких моделей, маленький экран.

Последний класс ноутбуков представлен моделями, ни в чем не уступающими настольным ПК. Данный класс получил название *DTR* (*Desktop Replacement — «замена настольного ПК»*). Данные ноутбуки имеют полный набор портов ввода-вывода, большие экраны и полноразмерные клавиатуры. Обратной стороной этих преимуществ являются большие габариты и масса, а также малое время работы от батарей.

Изготовители и поставщики ноутбуков для классификации используют некоторую комплексную оценку, четко определяющую **сегмент рынка**, для которого предназначена та или иная модель. Данная комплексная оценка основана на нескольких параметрах, важнейшими из которых являются массогабаритные характеристики, производительность и цена ноутбука. В соответствии с этой оценкой выделяются следующие классы:

- массовые модели;
- бизнес-модели;
- ультрапортативные модели;
- высокопроизводительные модели.

Заметим, что данная классификация очень похожа на классификацию по массе и габаритам, но не совпадает с ней. Например, с учетом цены «тонкие и легкие» ноутбуки могут быть как массовыми моделями, так и моделями бизнес-класса.

По **производительности** и **функциональным возможностям** ноутбуки различаются в зависимости, во-первых, от величины производительности, во-вторых, от того, какие подсистемы (графическая, звуковая, ввода-вывода) портативной ЭВМ и в какой степени развиты. Несмотря на огромный прогресс в области производства микроэлектронного оборудования для ноутбуков, все еще продолжает действовать правило «больше размеры — выше производительность и шире функциональные возможности».

Тип центрального процессора оказывает огромное влияние на габариты и производительность портативного ПК, поэтому нередко используется для класси-

фикации ноутбуков. Поскольку число производителей ЦП для ноутбуков на сегодняшний день весьма ограниченно, выделяют несколько классов.

1. Ноутбуки на базе серийных процессоров фирм AMD и Intel (на сегодняшний день это AMD Athlon 64 и Intel Pentium 4). Данные ЦП применяются лишь в ноутбуках DTR-класса. Они обеспечивают максимальную мощность, но, во-первых, быстро расходуют заряд батарей, во-вторых, нуждаются в мощной системе охлаждения.

2. Ноутбуки на базе мобильных процессоров фирмы AMD (на сегодняшний день это AMD Mobile Athlon XP) из-за сравнительной дешевизны процессора применяются в основном в массовых моделях и обеспечивают средний уровень производительности и энергопотребления.

3. Ноутбуки на базе мобильных процессоров фирмы Intel (на сегодняшний день это Intel Pentium M, платформа Centrino, первый ЦП, разработанный специально для портативных ПК) представлены практически во всех классах, за исключением высокопроизводительных моделей. Широкий спектр моделей с самым разным энергопотреблением и производительностью, а также активные маркетинговые усилия Intel по продвижению платформы Centrino обеспечивают данному классу безусловное лидерство на современном рынке.

4. Ноутбуки на базе процессоров VIA C3 и Trensmeta Efficeon характеризуются крайне низким энергопотреблением, и поэтому встречаются в классе ультрапортативных моделей.

Как уже было отмечено, архитектура ноутбуков имеет множество особенностей по сравнению с традиционными настольными ЭВМ. Разработка портативного ПК — это всегда поиск компромисса между взаимоисключающими требованиями, важнейшими из которых являются высокая производительность и низкое энергопотребление.

В связи с этим большинство комплектующих для ноутбуков (центральные процессоры, графические процессоры, жесткие диски) изготавливаются по особой технологии (или специально адаптируются) с целью снижения энергопотребления при сохранении производительности. Энергосбережение достигается, во-первых, за счет снижения напряжения питания, во-вторых, за счет широкого применения (вплоть до уровня ядра ЦП) алгоритмов динамического управления питанием. Данные технологические решения позволяют существенно увеличить время работы ноутбука от аккумуляторных батарей.

Остается добавить, что ведущими мировыми производителями ноутбуков, помимо таких значительных компаний, как IBM, HP, Dell, ASUS и Toshiba, на сегодняшний день являются крупные тайваньские фирмы — Quanta, Arima, Winstron, FIC, ECS, Clevo, Mitac и Compal.

Широкое внедрение ноутбуков во все сферы деятельности человека вызвало к жизни и потребность их дальнейшей миниатюризации. Это привело к появлению

карманных ПК (КПК). Особенно востребованными они оказались в сфере развлечений (поездки, полеты, отдых на курортах и др.).

КПК классифицируют (рис. 1.2) по разным признакам: тип ОС, тип дисплея, функциональные возможности и др.

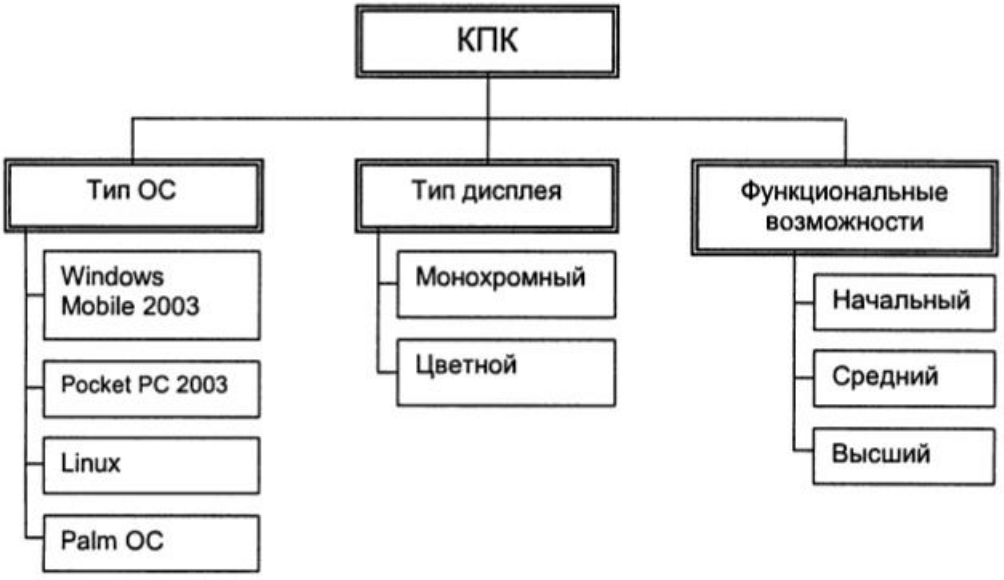


Рис. 1.2. Классификация КПК

КПК, как и остальные виды ПК, прошли несколько стадий своего развития.

На начальном уровне КПК представляли собой приборы небольшого размера, очень легкие и очень тонкие. Экран прикрывала темная крышка. На этом уровне в КПК отсутствовала возможность ввода рукописного текста, имелся ограниченный набор программ обработки, объем памяти составлял единицы Мбайт, питание от батареек формата AAA, имелась возможность подключения к настольному ПК со скоростью передачи данных до 56 Кбит/с. Примером таких КПК являются: Casio PV-s450, m500, m505.

Зарождением эпохи массовых КПК можно считать 1992 г., когда компания Apple представила модель Newton NotePad (в 1993 г. поступившую в продажу под названием Message Pad). Примерно в это же время шла разработка первого проекта от Palm Computing Inc. в союзе с Casio, Tandy, AOL, Intuit, Datalight. Характеристики КПК впечатляли: экран имел разрешение 320 × 256 точек, встроенная ОС была объектно-ориентированной, многозадачной, многопоточной, с поддержкой графического интерфейса. Главным недостатком, определившим неуспех модели, явилось отсутствие возможности установки программ сторонних производителей.

В 1995 г. Palm Computing была поглощена корпорацией U.S. Robotics, и в 1996 г. появился успешный КПК Palm Pilot 1000. В 1997 г. выходят Palm Pilot Personal и Professional со встроенной Palm OS.

Вплоть до 2001 г. устройства на базе Palm OS доминировали на рынке КПК. Рыночная доля Palm Computing составляла более двух третей. Вышедшая в 2000 г. Windows CE от Microsoft хотя и обладала более современной архитектурой, широкими возможностями, но предъявляла высокие требования к аппаратной платформе, что отрицательно сказывалось на цене. Такое положение дел не устраивало компанию Microsoft. Был предпринят ряд шагов по устранению недостатков КПК на базе Windows CE. Самый главный из них — высокая цена устройства — исчез после начала массового производства устройств несколькими крупными компаниями (например, Compaq). Функциональность этих моделей заметно превосходила КПК на базе Palm, и, начиная с 2002 г., Microsoft занимает ведущие позиции на рынке КПК.

Отметим, что архитектура КПК значительно отличается от архитектуры остальных классов ЭВМ. Основу процессоров КПК составляет архитектура ARM (Advanced RISC Machines), разработанная в 1983–1985 гг. в компании Acorn Computers. В 1990 г. Acorn, работавшая над развитием ARM уже в сотрудничестве с Apple, преобразовала подразделение, занимавшееся ARM, в отдельную фирму — Advanced RISC Machines. Руководство фирмы избрало оригинальный способ ведения бизнеса — позиционирование ARM в качестве «встраиваемого» вычислительного ядра, купив которое, любой желающий может интегрировать в свои специализированные процессоры. С этих пор архитектура ARM неуклонно развивается, являясь базой практически всех КПК, включая продукты главного производителя микропроцессоров — компании Intel.

1.2.1.3. Серверы

Слово **«сервер»** происходит от английского *serve* — обслуживать. Под **сервером** мы будем понимать выделенную ЭВМ (как правило, в составе вычислительной сети), обладающую некоторыми аппаратно-программными ресурсами и предоставляющую данные ресурсы пользователям по их запросам. Существует несколько типов серверов, ориентированных на разные применения: файл-сервер, сервер баз данных, принт-сервер, вычислительный сервер, сервер приложений. Таким образом, тип сервера определяется видом ресурса, которым он владеет (файловая система, базы данных, принтеры, процессоры или прикладные пакеты программ).

Серверы стали продуктом специализации персональных компьютеров в локальных сетях. Для эффективной работы сразу многих пользователей с данными целесообразно выделить в локальной сети один или несколько компьютеров, занятых выполнением специализированных задач, отвечающих запросам большинства пользователей. Такими задачами могли быть организация централизованного хранения файлов с предоставлением клиентам возможности считывать и записы-

вать файлы, обработка запросов к базам данных, организация очередей печати на общий принтер и т. п.

Возникла модель сетевой работы с данными, получившая название **«модель файл-сервера»**: данные находятся на сервере, обработка данных и их визуализация производятся на компьютере клиента. В этой модели дисковое пространство сервера является продолжением дискового пространства компьютера клиента, а сеть выступает как связующее звено файловых систем.

Однако этот подход имеет много недостатков: большой сетевой трафик, неэффективность использования ресурсов сервера, примитивная обработка транзакций, нагрузка на компьютер клиента и т. д. Поэтому возникла альтернативная архитектура, получившая название **«клиент-сервер»**: данные хранятся и обрабатываются на сервере, визуализация производится на компьютере клиента. При этом громоздкое и дорогое ПО исполняется на сервере, который в этом случае называется *сервером приложений*. На компьютере клиента остается только ПО, предназначенное для организации связи с сервером, формирования запросов и визуализации данных.

Две модели архитектуры обусловили появление аппаратных серверов двух крайних типов: оптимизированных на работу с файлами и на проведение вычислений. Первому типу соответствуют устройства со сравнительно слабым процессором, но развитой дисковой и сетевой подсистемами, нацеленными на быструю выдачу практически готовых данных; второй тип может иметь много мощных процессоров и способен выполнить большой объем вычислений в сжатые сроки.

С другой стороны, существует классификация серверов, определяющаяся масштабом сети, в которой они используются: **сервер рабочей группы, сервер отдела** или **сервер масштаба предприятия (корпоративный сервер)**. Эта классификация весьма условна. Например, размер группы может меняться в диапазоне от нескольких человек до нескольких сотен человек, а сервер отдела обслуживать от 20 до сотен пользователей. Очевидно, в зависимости от числа пользователей и характера решаемых ими задач, требования к составу оборудования и программного обеспечения сервера, к его надежности и производительности сильно варьируются. Кроме этого, деление серверов согласно приведенной классификации является сиюминутным — серверы, находящиеся сегодня на уровне предприятия, через год-два не смогут претендовать на уровень отдела. Тем не менее данная классификация достаточно полезна, поскольку интегрирует в себя много различных параметров, основной из которых — производительность.

Производительность сервера определяется производительностью и сбалансированностью его трех основных подсистем: процессорной, дисковой и сетевой. Как правило, для серверов характерны минимальные требования к производительности мультимедийных компонент — графической и аудио подсистем. Важным для серверов является оснащение достаточно мощными блоками питания для воз-

можности установки дополнительных плат расширения и дисковых накопителей, а также устройствами бесперебойного питания.

В силу характера решаемых задач производительность процессорной подсистемы сервера зависит не столько от мощности используемых процессоров, сколько от их количества, объема кэш-памяти (имеющейся внутри процессора и внешней), типа и объема оперативной памяти, а также от быстродействия шины и набора микросхем системной логики. Процессоры для серверов имеют, как правило, укороченный конвейер (для более быстрого переключения контекстов) и увеличенный объем внутренней кэш-памяти. Данные процессоры, такие, как Intel Xeon или AMD Opteron, составляют отдельный сегмент рынка микропроцессоров.

В настоящее время однопроцессорные ЭВМ используются лишь в качестве серверов начального уровня. Начиная с уровня отдела, применяются двух-, четырех- и восьмипроцессорные конфигурации, а уровень предприятия представлен серверами, содержащими несколько десятков процессоров. Таким образом, в большинстве своем серверы являются многопроцессорными вычислительными системами. Память у всех процессоров общая, построенная на основе архитектур SMP или NUMA, сервер управляется одной операционной системой. Более подробно данные архитектуры будут рассмотрены в главе 5 «Параллельные ЭВМ и ВС» настоящего учебника.

Быстродействие дисковой подсистемы определяется используемыми жесткими дисками, контроллером и применяемым интерфейсом. Обычно в серверах устанавливаются SCSI-диски, обеспечивающие высокую эффективность передачи данных. Их сильная сторона — способность выстраивать запросы на обслуживание в очередь и выполнять их таким способом, который оптимизирует перемещение головок по цилиндрам, что в среднем приводит к более быстрому обмену данными. Кроме того, шины SCSI позволяют подключить к системе большее число дисков — в сравнении с IDE-стандартом, широко распространенным в настольных системах. Подробнее о стандартах шин см. раздел 4.4 настоящего учебника.

В серверах широко применяется организация дисков в RAID-массивы, что повышает производительность подсистемы, ее отказоустойчивость или и то и другое.

Третьей важнейшей компонентой серверов является сетевая подсистема. В зависимости от уровня сервера используются различные типы и количество сетевых адаптеров. В серверах рабочих групп и отдела применяются 1–2 адаптера массового производства (такие, как Fast Ethernet или GigaEthernet). Сервер уровня предприятия содержит несколько высокопроизводительных сетевых адаптеров средней серийности, таких, как Myrinet или Infiniband.

Одна из основных тенденций в развитии серверостроения — размещение все большей вычислительной мощности во все меньшем объеме. При этом подсисте-

ма хранения данных выделяется в отдельное устройство, связанное с сервером высокопроизводительной сетью передачи данных. Оказалось гораздо удобнее сосредоточить множество таких устройств в специально выделенном помещении, как правило, очень небольшой площади и смонтировать их в стандартную 19-дюймовую стойку. Многие производители выпускают серверы, которые по выбору покупателя могут трансформироваться из стоечного в пьедестальное исполнение, и наоборот.

Следующий шаг в этом направлении — появление «*серверов-лезвий*» (blade), еще больше увеличивающих удельную вычислительную мощность. Это достигается благодаря переходу к компактным одноплатным серверам, вставляемым в общее шасси, которое монтируется в стандартную 19-дюймовую стойку. Кроме выгод, связанных с экономией площадей, подобный подход несколько облегчает проблему охлаждения, так как серверы в шасси располагаются вертикально и появляется возможность использовать конвективные потоки воздуха.

На плате располагаются до четырех процессоров, память, набор микросхем и один или два жестких диска. Общаются между собой серверы по шинам, расположенным на объединительной плате. Обычно используются такие интерфейсы, как Gigabit Ethernet, FibreChannel и Myrinet. В шасси в зависимости от компоновки может входить до 16 одноплатных компьютеров, один или два из которых выполняют функции управления системой. По мере роста потребностей в вычислительной мощности в шасси можно вставлять дополнительные серверы.

1.2.1.4. Мэйнфреймы

Мэйнфрейм — это синоним понятия «*большая универсальная ЭВМ*». Слово «мэйнфрейм» появилось в 70–80 е годы XX в., когда большие универсальные ЭВМ состояли из центрального процессора и множества подсистем: дисковых, ленточных, терминальных контроллеров и т.п. Практически все фирмы, которые производили подобную технику (основная доля рынка приходилась на продукцию IBM, Hitachi, Amdahl), придерживались примерно одинаковой архитектуры. Каждая из подсистем ЭВМ располагалась в отдельной стойке. Отсюда и название «мэйнфрейм» (**main frame** — «главная стойка»); оно закрепилось за центральной стойкой, где находился процессор.

Мэйнфреймы и до сегодняшнего дня остаются наиболее мощными (не считая супер-ЭВМ) вычислительными системами общего назначения, обеспечивающими непрерывный круглосуточный режим эксплуатации. Они могут включать один или несколько процессоров, каждый из которых, в свою очередь, может оснащаться векторными сопроцессорами. Прогресс в области элементно-конструкторской базы позволил существенно сократить габариты основных устройств. Наряду со сверхмощными мэйнфреймами, требующими для охлаждения принудительной воздушной вентиляции, существуют модели, построенные по блочно-модульному принципу и не требующие специальных помещений и кондиционеров.

Основными поставщиками мэйнфреймов являются известные компьютерные компании IBM, Amdahl, ICL, Siemens Nixdorf и некоторые другие, но ведущая роль, безусловно, принадлежит компании IBM. Именно архитектура системы IBM/360, выпущенной в 1964 г., и ее последующие поколения стали образцом для подражания. В нашей стране в течение многих лет выпускались машины серии ЕС ЭВМ, являвшиеся отечественным аналогом этой системы.

В архитектурном плане мэйнфреймы представляют собой, как правило, многопроцессорные системы, содержащие один или несколько центральных и периферийных процессоров с общей памятью, связанных между собой высокоскоростными магистралями передачи данных. При этом основная вычислительная нагрузка ложится на центральные процессоры, а периферийные процессоры обеспечивают работу с широкой номенклатурой периферийных устройств.

Основная ориентация мэйнфреймов — *централизованная модель вычислений* и работа под управлением патентованных операционных систем, основное назначение — *непрерывное обслуживание интенсивного потока транзакций*. Первоначально мэйнфреймы имели ограниченные возможности для объединения в единую систему с использованием оборудования различных фирм-поставщиков. Однако повышенный интерес потребителей к открытым системам, построенным на базе международных стандартов и позволяющим достаточно эффективно использовать все преимущества такого подхода, заставил поставщиков мэйнфреймов существенно расширить возможности своих операционных систем в направлении совместимости. В настоящее время они демонстрирует свою «открытость», обеспечивая соответствие со спецификациями POSIX 1003.3, возможность использования протоколов межсоединений OSI и TCP/IP или предоставляя возможность работы на своих компьютерах под управлением операционной системы UNIX собственной разработки.

Стремительный рост производительности персональных компьютеров, рабочих станций и серверов создал тенденцию перехода с больших универсальных ЭВМ на компьютеры менее дорогих классов: рабочие станции и многопроцессорные серверы. Эта тенденция получила название «разукрупнение» (downsizing). Однако этот процесс в самое последнее время несколько замедлился. Основной причиной возрождения интереса к мэйнфреймам эксперты считают сложность перехода к распределенной архитектуре «клиент-сервер», которая оказалась выше, чем предполагалось.

Основным недостатком мэйнфреймов является относительно невысокое значение показателя «производительность/стоимость». Для распределенных сред, состоящих из многопроцессорных серверов и рабочих станций, данный показатель значительно выше. Но при этом следует помнить, что мэйнфреймы обеспечивают высочайший уровень надежности, готовности и информационной безопасности, достижение, и особенно поддержание которого в распределенных средах, требует значительных дополнительных затрат. При соблюдении высочайших требований по надежности, готовности и информационной безопасности мэйнфреймы имеют

самую низкую *стоимость владения* по сравнению с другими архитектурами, что практически компенсирует высокие первоначальные затраты на приобретение системы.

Таким образом, основная область применения мэйнфреймов — высоконадежная централизованная высокопроизводительная обработка массированного потока транзакций. Поэтому основными потребителями подобной техники являются крупные компании — банковские, промышленные, транспортные и т.п., выбор которых, помимо рассмотренных аспектов, определяет важнейшее преимущество мэйнфреймов перед остальными вычислительными системами — *более чем сорокалетняя преемственность систем*. Примером служит архитектурная линия IBM — S/360, S/370, S/390, которая постоянно развивается. Модели ЭВМ сменяют одна другую, а накопленное программное обеспечение продолжает успешно функционировать.

К примеру, крупнейшая американская железнодорожная компания Union Pacific работает на приложениях, которые были разработаны в 60-е и 70-е годы прошлого века. Поскольку за прошедшие годы ни география, ни инфраструктура компании принципиально не поменялись, специалисты Union Pacific просто наращивают созданную систему, обновляя модели используемых мэйнфреймов.

В мире существует огромная инсталлированная база мэйнфреймов, на которой работают десятки тысяч прикладных программных систем. Отказаться от годами наработанного программного обеспечения просто неразумно. Поэтому не наблюдается снижения продаж больших универсальных ЭВМ. Эти системы, с одной стороны, позволяют модернизировать существующие системы, обеспечив сокращение эксплуатационных расходов, с другой стороны, создают новую базу для наиболее ответственных приложений.

1.2.1.5. Супер-ЭВМ

Согласно ГОСТ 15971-90 под **супер-ЭВМ (суперкопьютером)** понимается ЭВМ, относящаяся к классу ВМ, имеющих самую высокую производительность, которая может быть достигнута на данном этапе развития технологии, и в основном предназначенных для решения сложных научно-технических задач. Суперкомпьютер — это ЭВМ мелкосерийного или штучного выпуска, многократно превосходящая по вычислительной мощности массово выпускаемые компьютеры. Конкретная, численно выраженная грань быстродействия, отделяющая супер-ЭВМ от остальных ЭВМ, довольно размыта и быстро смещается по мере непрерывного прогресса в области производства процессоров. Поэтому отличительными признаками супер-ЭВМ будем полагать следующие:

- супер-ЭВМ не является изделием массового выпуска, и, следовательно, при ее изготовлении и применении используются уникальные технологии, более дорогие и, возможно, менее удобные, чем массовые технологии;
- суперкомпьютер — это ЭВМ, ориентированная на *вычисления*, на заметное, минимум на порядок, снижение времени выполнения сложных расчетов по сравнению с персональными ЭВМ или рабочими станциями.

Именно ориентация на вычисления является главным отличительным признаком супер-ЭВМ от мэйнфреймов, основным назначением которых является обработка транзакций.

Понятие «супер-ЭВМ» появилось одновременно с началом серийного производства компьютеров. С этого времени для решения особо сложных, стратегических задач стали применяться самые передовые технологии, которые в силу своей сложности и дороговизны не были конвейерными. Примерно до середины 80-х годов XX в. речь шла почти исключительно о технологиях производства процессоров, и с высокой степенью точности можно было утверждать, что супер-ЭВМ — это ЭВМ, оснащенная специальным, особо мощным (и потому — особо дорогим) процессором.

Ситуация изменилась, когда в 1989 г. фирма Intel выпустила однокристальный микропроцессор i860, сравнимый по производительности с еще действовавшей в то время супер-ЭВМ Cray-1. Стало ясно, что с этого времени все процессоры будут только однокристальными, и это означает завершение 30-летней эпохи особо мощных и дорогих, изготавливаемых из отдельных элементов процессоров. Любую технологию повышения быстродействия процессора стало возможно реализовать на одном кристалле и за короткое время довести до крупносерийного выпуска. Поскольку производство микропроцессоров по существу является конвейерным и довольно дешевым, процессоры перестали делиться на «массовые и дешевые» и «особо мощные и дорогие».

Для создания супер-ЭВМ в этих условиях остался единственный путь — объединение многих процессоров для параллельного решения одной особо сложной задачи. С этого момента любая супер-ЭВМ обязательно стала представлять собой *параллельную многопроцессорную вычислительную систему*. Поэтому более подробно архитектура и структура супер-ЭВМ будут рассмотрены в главе 5 «Параллельные ЭВМ и ВС» настоящего учебника.

1.2.2. Классификация вычислительных систем

Вычислительные системы классифицируются (рис. 1.3) по

- назначению;
- сложности;
- типу используемых ЭВМ;
- степени централизации управления;
- принципу закрепления вычислительных функций;
- формам обслуживания;
- временному режиму работы;
- степени территориальной разобщенности;
- стоимости и объему внешней памяти;

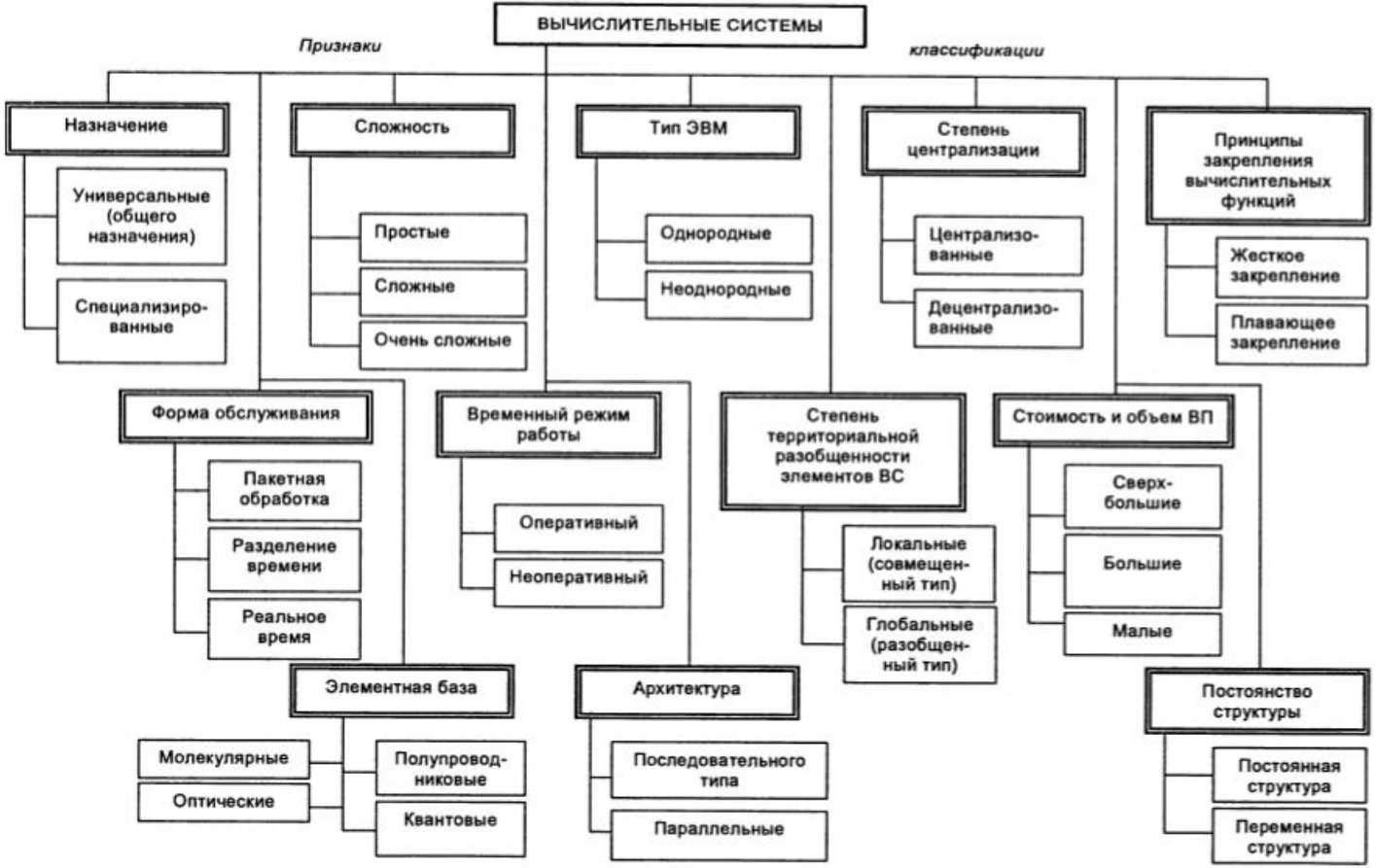


Рис. 1.3. Классификация вычислительных систем

- элементной базе;
- архитектуре;
- постоянству структуры.

Рассмотрим эти признаки классификации несколько подробнее.

По назначению ВС делятся на *универсальные* (общего назначения) и *специализированные*. Универсальные ВС используются для решения широкого круга задач различного характера, а специализированные— определенного круга задач. Специализация ВС может определяться не только характером решаемых задач и ограниченностью их перечня, но и наличием в системе элементов (устройств) специального назначения.

По сложности (в зависимости от количества и сложности входящих в них элементов) различают *простые*, *сложные* и *очень сложные* ВС.

Простые ВС — это системы, которые не имеют большого количества элементов и связей. Элементы выполняют простые функции. Они легко описываются и анализируются.

Сложные ВС — системы, обладающие разветвленной структурой, большим количеством взаимосвязанных и зависимых элементов, которые выполняют сложные функции. Как правило, сложные ВС являются многоцелевыми системами. Для них могут быть проанализированы и описаны лишь отдельные состояния.

Очень сложные ВС — системы, которые ввиду их сложности трудно подробно и точно описать.

По типу ЭВМ, из которых комплектуется ВС, различают *однородные* и *неоднородные* ВС. Принадлежность ВС к однородной или неоднородной системе определяется не только ЭВМ, но и их программной совместимостью. Степень программной совместимости ЭВМ тем выше, чем выше степень их сходства по таким параметрам, как адресность, структура команд, представление данных и др.

По степени централизации ВС разделяют на *централизованные* и *децентрализованные*. При полной централизации все функции управления ВС сосредоточены в одном элементе. В качестве управляющего органа выступает ЭВМ-диспетчер, координирующая загрузку машин и их взаимодействие в процессе выполнения задания. При таком способе управления достигается высокий коэффициент использования ЭВМ в ВС. Однако при отказе управляющей ЭВМ вся ВС выходит из строя. В децентрализованных ВС функции управления распределены между ЭВМ системы. Структура управления обычно является иерархической, при этом может быть несколько уровней управления.

По принципу закрепления вычислительных функций за отдельными ЭВМ различают ВС с *жестким* и *плавающим* закреплением функций.

По формам обслуживания ВС можно разделить на системы с *пакетной обработкой*, с *разделением времени* и *реального времени*. В ВС с пакетной обработкой пользователи, как правило, не имеют непосредственного доступа к ЭВМ. Заранее подготовленные задания накапливаются в очереди и затем выполняются. В системах с разделением времени (СРВ) каждый пользователь имеет непосредственный доступ к ЭВМ или ВС через терминал (дисплей, ПЭВМ, рабочая станция и др.). В этом режиме можно реализовать и пакетную обработку, если в этом есть необходимость. В системах реального времени ВС с помощью датчиков соединена с источником воздействий, непосредственно реагирующим на получаемые сигналы. Такие системы используют в системах автоматизации технологическими процессами, управления движением судов, ракет, самолетов и др.

По временному режиму работы различают ВС, работающие в *оперативном* и *неоперативном* режиме. ВС, работающие в оперативном режиме, являются системами реального времени, роль человека в них сведена к минимуму. Системы, работающие в неоперативном режиме, называют иногда информационно-логическими системами (ИЛС) или информационно-вычислительными системами (комплексами) (ИВС, ИВК). Роль человека в таких системах выше. Его вмешательство может прерывать процесс обработки информации.

По степени территориальной разобщенности основных элементов ВС различают системы *совмещенного типа,* или *локальные,* и системы *разобщенного типа,* или *глобальные*. Локальными ВС считают такие, в которых время передачи информации от ЭВМ к ЭВМ мало по сравнению со временем выполнения задания на одной из машин. ЭВМ такой системы расположены, как правило, в пределах одного здания.

ВС разобщенного типа, или глобальные, характерны тем, что время передачи информации по каналам связи соизмеримо с временем выполнения задания. ЭВМ такой ВС находятся на значительном удалении друг от друга.

По стоимости и объему внешней памяти ВС можно разделить на *сверхбольшие*, *большие* и *малые*. **По элементной базе** ВС можно разделить на *полупроводниковые*, *квантовые*, *молекулярные*, *оптические* и др. К полупроводниковым ВС относят традиционные системы, построенные на полупроводниковых интегральных схемах. Их, в свою очередь, можно классифицировать на более мелкие подсистемы в зависимости от технологии изготовления элементной базы, заложенных физических принципов и др.

В *квантовых* ВС предпринимается попытка реализации ЭВМ на базе квантовых эффектов. В *молекулярных* ВС предполагается использовать в качестве элементной базы молекулы с резонансной туннельной проводимостью.

Оптические ВС предполагают использование в качестве элементной базы различных источников излучения как в видимой части спектра, так и невидимой.

По архитектуре ВС можно разделить на ВС *последовательного типа* (обычные ЭВМ) и *параллельные* ВС. Архитектура параллельных ВС настолько

разнообразна, что требует отдельной классификации. Учитывая возрастающую важность параллельных ВС, их классификация будет рассмотрена отдельно в соответствующем разделе.

По постоянству структуры системы разделяются на ВС с *постоянной* и с *переменной* структурой. ЭВМ в ВС между собою могут связываться по информации (по данным) и по управлению. Система относится к ВС с постоянной структурой, если в процессе ее функционирования состав функциональных и управляющих связей и их ориентация остаются неизменными. В противном случае система называется ВС с переменной структурой.

1.3. ПОКАЗАТЕЛИ КАЧЕСТВА ФУНКЦИОНИРОВАНИЯ ЭВМ И ВС

Качеством (***quality***) будем называть совокупность свойств, определяющих возможность использования ВС по назначению. Качество ВС определяется ее техническими, эксплуатационными, экономическими и эргономическими характеристиками. Его можно оценить численно с помощью *показателей качества*, которые должны иметь физический смысл, достаточно полно описывать качество, просто вычисляться и не противоречить друг другу.

Показатель качества — это вектор, компонентами которого служат показатели свойств, являющиеся частными показателями качества.

Показатели качества можно классифицировать по *пригодности*, *оптимальности* и *превосходству*. ВС, удовлетворяющие показателям **пригодности**, обладают одинаковым качеством; удовлетворяющие показателям **оптимальности** являются наилучшими, т. е. обладают наивысшим качеством; удовлетворяющие показателям **превосходства** считаются превосходящими по качеству остальные системы.

Показатели качества удобно делить также на *интервальные*, *интегральные* и *точечные* (локальные). **Интервальные** показатели дают возможность оценить изменение качества во времени за определенный его промежуток, **интегральные** — позволяют оценить качество в среднем при функционировании ВС в течение длительного времени, **точечные** — характеризуют качество в данный момент времени.

При оценке качества ВС в процессе ее разработки или эксплуатации встречаются с двумя трудностями. Первая заключается в том, что в большинстве случаев не удается установить единый обобщенный показатель качества, который позволил бы сравнить различные системы и выбрать наилучшую. Это объясняется тем, что качество оценивается множеством свойств.

Вторая трудность определяется тем, что пока не существует методики установления требований на показатели качества. Из-за этого пока что не удается разработать критерий оптимальности системы в смысле ее качества на все случаи жизненного цикла системы.

Основные показатели, характеризующие качество ЭВМ или ВС, можно объединить в следующие группы:

1) показатели эффективности;
2) показатели производительности;
3) показатели надежности и готовности;
4) показатели адаптивности;
5) показатели экономичности.

Рассмотрим более подробно перечисленные группы.

1.4. ЭФФЕКТИВНОСТЬ ЭВМ И ВС

Эффективность ЭВМ или ВС (**System Efficiency**) — это качество системы, характеризующее ее техническое совершенство, экономическую целесообразность и отражающее степень ее соответствия своему назначению.

Эффективность W ЭВМ или ВС зависит от множества параметров входящих потоков требований $G_п$ (или параметров реализуемых алгоритмов), множества параметров системы $G_с$, множества параметров элементов системы $G_э$, множества параметров $G_у$, характеризующих влияние внешних факторов на процесс функционирования ВС.

Другими словами, W есть функционал от процесса функционирования системы. Так как ЭВМ или ВС функционируют в условиях воздействия случайных факторов, то значениями функционала являются случайные величины. При этом на практике весьма затруднительно формально определить множества $G_п$, $G_с$, $G_э$, $G_у$. Этот факт делает невозможным и нецелесообразным расчет эффективности ЭВМ и ВС в виде единственного количественного показателя $W(G_п, G_с, G_э, G_у)$.

Поэтому на практике под эффективностью понимается *способность ВС выполнять целевые задачи, для решения которых она предназначена.* Показателем эффективности (ПЭ) называется численная величина, количественно отражающая качество решения задач какой-либо вычислительной системой. В качестве ПЭ используют показатель, отражающий вероятность решения поставленной задачи $W_i = P(A_i)$, где $A_i = f(t)$ — событие, заключающееся в решении поставленной задачи.

Показатели эффективности можно разделить на *технические* и *экономические*.

Наиболее рациональным показателем технической эффективности является **вероятность $P(t, t_1)$ правильного решения задачи**, поступившей в момент времени t_1, за время, не превышающее t. Этот показатель достаточно полно характеризует способность ЭВМ или ВС решать задачи и зависит от многих технических характеристик системы и показателей качества: надежности, ремонтопригодности, производительности, объема памяти и ряда других характеристик. Кроме этого, показатель $P(t, t_1)$ определяется характеристиками и приоритетностью потока задач. Поэтому он может быть обобщенным показателем качества ВС лишь при заданных потоках заявок и приоритетности их обслуживания, в противном же случае является показателем качества системы, состоящей из источника потока заявок и ВС.

1.5. ПРОИЗВОДИТЕЛЬНОСТЬ ЭВМ И ВС

1.5.1. Понятие производительности

Производительность (быстродействие, performance) является одной из важнейших характеристик системы. Однако, к сожалению, до настоящего времени вопрос о том, что же такое производительность или быстродействие системы, остается открытым. Это связано с тем, что даже на оценку производительности одной ЭВМ влияют слишком много факторов.

Среди таких факторов можно отметить следующие:

- тип задач;
- число тех или иных операций, выполняемых при решении задачи;
- стиль программирования и другие особенности программы;
- логические возможности системы команд;
- структура процессора;
- характеристики и организация оперативной (ОП) и внешней памяти (ВП);
- особенности системы ввода-вывода;
- состав и характеристики УВВ и др.

В настоящее время существует целый ряд единиц измерения производительности ЭВМ и ВС. Рассмотрим наиболее употребительные из них.

Одной из единиц измерения производительности процессора (по отношению к времени выполнения) является **MIPS (Million Instructions Per Second** — «миллион команд в секунду»). В настоящее время имеется несколько различных вариантов интерпретации определения MIPS.

В общем случае MIPS есть скорость выполнения операций в единицу времени, т. е. для любой данной программы MIPS есть просто отношение количества выполненных команд в программе к времени ее выполнения. Таким образом, производительность может быть определена как обратная к времени выполнения величина, причем более быстрые машины при этом будут иметь более высокое значение MIPS.

Положительными сторонами этой единицы измерения является то, что ее легко понять, так как более быстрая машина характеризуется большим значением MIPS. Однако использование MIPS в качестве метрики для сравнения различных ЭВМ наталкивается на три проблемы. Во-первых, MIPS зависит от набора команд процессора, что затрудняет сравнение по MIPS компьютеров, имеющих разные системы команд. Во-вторых, MIPS даже на одном и том же компьютере меняется от программы к программе. В-третьих, MIPS может меняться по отношению к производительности в противоположную сторону.

Классическим примером для последнего случая является значение MIPS для машины, в состав которой входит сопроцессор плавающей точки. Поскольку в общем случае на каждую команду с плавающей точкой требуется большее коли-

чество тактов синхронизации, чем на целочисленную команду, то программы, используя сопроцессор плавающей точки вместо соответствующих подпрограмм из состава программного обеспечения, выполняются за меньшее время, но имеют меньшее значение MIPS. При отсутствии сопроцессора операции над числами с плавающей точкой реализуются с помощью подпрограмм, использующих более простые команды целочисленной арифметики и, как следствие, такие машины имеют более высокие значение MIPS, но выполняют настолько большее количество команд, что общее время выполнения значительно увеличивается.

Одно из определений MIPS связано с очень популярным в свое время компьютером VAX 11/780 компании DEC. Именно эта ЭВМ была принята в качестве эталона для сравнения производительности различных машин.

Считалось, что производительность VAX 11/780 равна 1 MIPS (одному миллиону команд в секунду). Еще одно определение «одного MIPS» связано с машиной IBM RS/6000. Дело в том, что ряд производителей и пользователей (последователей фирмы IBM) предпочитают сравнивать производительность своих компьютеров с производительностью современных компьютеров IBM, а не со старой машиной компании DEC. 1 MIPS по этому определению равен примерно 1.6 MIPS для машин серии VAX.

Из вышеизложенного следует, что для определения производительности величина MIPS является весьма относительной и не очень объективной. Учитывая этот факт, а также чрезвычайную важность научно-технических расчетов, активно использующих арифметику с плавающей точкой, для оценки производительности ЭВМ и ВС была введена величина **MFLOPS (Million FLoating point Operations Per Second** — «миллион операций с плавающей точкой в секунду»); читается «мегафлопс». Соответственно, существуют величины GFLOPS («гигафлопс», 1000 MFLOPS), TFLOPS («терафлопс», 1000 GFLOPS) и PFLOPS («петафлопс», 1000 TFLOPS).

Как единица измерения, MFLOPS, предназначена для оценки производительности только операций с плавающей точкой и поэтому не применима вне этой ограниченной области. Например, программы компиляторов имеют значение MFLOPS, близкое к нулю, вне зависимости от того, насколько быстра машина, поскольку компиляторы редко используют арифметику с плавающей точкой.

Величина MFLOPS базируется на количестве выполняемых операций, а не на количестве выполняемых команд. По мнению многих программистов, одна и та же программа, работающая на различных ЭВМ, будет выполнять разное количество команд, но одно и то же количество операций с плавающей точкой. Именно поэтому MFLOPS предназначается для более объективного сравнения различных машин между собой. Но при использовании этой величины следует иметь в виду, что наборы операций с плавающей точкой не совместимы на различных компьютерах. Например, в суперкомпьютерах фирмы Cray Research отсутствует команда деления (имеется, правда, операция вычисления обратной величины чис-

ла с плавающей точкой, а операция деления может быть реализована с помощью умножения делимого на обратную величину делителя). В то же время многие современные микропроцессоры имеют команды деления, вычисления квадратного корня, синуса и косинуса. Кроме того, следует учитывать тот факт, что величина MFLOPS меняется не только на смеси целочисленных операций и операций с плавающей точкой, но и на смеси быстрых и медленных операций с плавающей точкой. Например, программа со 100% операций сложения будет иметь более высокое значение, чем программа со 100% операций деления.

Однако, как и любая другая единица измерения, величина MFLOPS для отдельной программы не может быть обобщена на все случаи жизни, чтобы представлять единственную единицу измерения производительности компьютера, хотя очень соблазнительно характеризовать машину единственным значением MIPS или MFLOPS без указания программы.

На практике чаще всего используют понятия **пиковой** (технической) и **реальной** производительности. Кроме этих двух показателей, производительность ЭВМ или ВС характеризуется **временем ответа**.

1.5.2. Пиковая производительность

Пиковая, или **техническая,** производительность представляет собой *теоретический максимум быстродействия компьютера при идеальных условиях*. Идеальные условия обеспечиваются при:

1) подаче на вход процессоров ВС независимых друг от друга идеальных программ, каждая из которых состоит из бесконечной последовательности не связанных между собой и не конфликтующих при доступе в память команд (т. е. когда результат любой операции не зависит от действий, выполненных другими командами);

2) задействовании в процессе выполнения идеальной программы всех арифметико-логических устройств (АЛУ) всех процессоров, входящих в состав ВС. При этом предполагается, что все операнды выбираются из наиболее быстрой памяти данных (например, внутрикристальной кэш-памяти данных), а команды — из наиболее быстрой памяти команд в общей иерархии памяти ВС.

Разумеется, идеальные условия чисто гипотетические, и на практике ни одна система не в состоянии работать сколь-нибудь длительное время с пиковой производительностью, хотя и может приближаться к этой величине. Например, на тестах Linpack при больших размерностях обрабатываемых матриц (1000 × 1000) многие ЭВМ и ВС показывают производительность в диапазоне от 0,8 до 0,95 пикового значения.

Пиковая производительность является единственной по-настоящему объективной величиной (для ее определения необходимо знать всего несколько параметров ВС) и совершенно не зависит от типа выполняемой программы. Для подавляющего большинства современных ЭВМ темп формирования результатов на

выходе арифметического конвейера определяется тактовой частотой. Зная число арифметических конвейеров в процессоре и число процессоров в ВС, нетрудно установить соответствие между единицами измерения тактовой частоты и производительности ВС. Очевидно, что 1 МГц соответствует 1 MFLOPS или 1 MIPS пиковой производительности одного конвейера в зависимости от того, какой тип операций рассматривается — с плавающей или с фиксированной точкой. Не следует забывать также о разрядности обрабатываемых чисел, чтобы избежать некорректного сравнения производительности для 32- и 64-разрядных данных. Чтобы определить пиковую производительность ВС, надо умножить тактовую частоту на количество параллельно выполняемых операций. Например, арифметическое устройство микропроцессора Pentium за каждый такт может формировать один результат полноразрядной (64 бит) операции с плавающей точкой или два результата 32-разрядных целочисленных операций. Следовательно, для Pentium/90 пиковая производительность равна 90 MFLOPS при выполнении вычислений с плавающей точкой и 180 MIPS при целочисленной 32-разрядной обработке. Умножив эти значения на число процессоров в ВС, получим ее пиковую производительность.

Значения пиковой производительности при всей простоте их получения могут оказаться чрезвычайно полезными при обсуждении достоинств компьютеров. Особенно удобно использовать показатели предельного быстродействия для сравнения возможностей ЭВМ в первом приближении. Однако сравнение ЭВМ или ВС с использованием значений пиковой производительности не всегда приводит к адекватным результатам. Разные форматы команд разных ЭВМ имеют разную длительность исполнения, и время исполнения последовательности команд зависит от конкретного места каждой команды в последовательности. Кроме этого, на результат влияет разрядность обрабатываемых чисел, поэтому не совсем корректно сравнивать пиковую производительность 32-разрядных и 64-разрядных ЭВМ.

1.5.3. Реальная производительность

При выполнении реальных прикладных программ **реальная (эффективная, номинальная)** производительность может весьма существенно (до нескольких раз) отличаться от пиковой. Реальная производительность зависит от архитектуры ЭВМ и ВС, от программы, обрабатываемых данных.

В настоящее время для оценки реальной производительности ЭВМ и ВС используют целый ряд различных методик. Рассмотрим наиболее используемые из них.

Очень часто для оценки производительности как отдельных ЭВМ, так и ВС широко используют наборы задач (тесты) для той или иной области их применения. Время выполнения каждой из задач набора составляет основу для расчета *индекса производительности* исследуемой ВС. Этот индекс является относительной оценкой, несущей информацию о том, насколько быстрее или медленнее ис-

следуемая ВС выполняет подобные задачи по сравнению с некоторой широко распространенной ЭВМ, называемой *базовой*, или *эталонной*. Очевидно, что если мы умеем определять производительность эталонной ЭВМ, то не составит труда перейти от индекса производительности к абсолютным ее значениям для исследуемой ВС.

Следует заметить, что при оценке производительности на тестах приходится решать три проблемы, связанные с анализом результатов контрольного тестирования производительности:

- отделение показателей, которым можно доверять безоговорочно, от тех, которые должны восприниматься с известной долей настороженности (проблема достоверности оценок);
- выбор контрольно-оценочных тестов, наиболее точно характеризующих производительность при обработке типовых задач пользователя (проблема адекватности оценок);
- правильное истолкование результатов тестирования производительности, особенно если они выражены в нестандартных единицах измерения (проблема интерпретации).

Различают следующие группы тестов:

1) тесты производителей;

2) стандартные тесты;

3) тесты пользователей.

Первую группу тестов (**тесты производителей**) разрабатывают компании — изготовители ЭВМ или ВС для «внутреннего» применения — оценки качества собственных продуктов. Главная особенность данных тестов заключается в том, что они ориентированы на сравнение ограниченного множества однотипных компьютеров, часто относящихся к одному семейству.

Например, для оценки производительности микропроцессоров с архитектурой x86 компания Intel в 1992 г. предложила индекс производительности iCOMP (Intel Comparative Microprocessor Performance). В качестве эталонного процессора принята модель 486 SX-25, значение индекса для которого равно 100. Индекс iCOMP определяется при выполнении смеси операций, состоящей из 67% операций над 16-разрядными целыми числами, 3% операций над 16-разрядными числами с плавающей точкой, 25% — над 32-разрядными целыми числами и 5% — над 32-разрядными числами с плавающей точкой. К примеру, индексы iCOMP для микропроцессоров 486 SX2-50, Pentium-100 и Pentium-166 равны 180, 815 и 1308 соответственно. Следует отметить, что индекс iCOMP оценивает производительность микропроцессора как такового, а не вычислительной системы, включающей еще оперативную память и внешние устройства.

В фирме IBM имеются специализированные тестовые пакеты LSPR для измерения производительности ЭВМ семейств System/370 и System/390, а также тесты RAMP-C для компьютеров AS/400.

Тесты производителей являются почти идеальным средством оценивания быстродействия систем, для которых они созданы (действительно, кто знает структурные особенности машины лучше ее создателя), однако вряд ли могут быть в чистом виде использованы для других компьютеров — сказывается слишком явная ориентация тестов на конкретную «фирменную» архитектуру.

Вторую группу составляют **стандартные тесты**, разработанные для сравнения широкого спектра ЭВМ и ВС. Тесты этой категории — продукт деятельности независимых аналитиков (например, Джека Донгарры, предложившего тестовый пакет Linpack) или групп, объединяющих крупнейших производителей компьютеров (SPEC, TPC-C, DevectClub), что практически исключает возможность ориентации стандартного теста на конкретного поставщика. Например, для оценки серверов, обрабатывающих транзакции в реальном времени в OLTP-системах, используется тестовый набор Transaction Processing Performance Council (TPC-C). Производительность серверов оценивается числом транзакций, исполняемых за минуту (tpmC).

Тесты третьей группы — **пользовательские тесты** — создаются крупными компаниями, специализирующимися на внедрении компьютерных технологий, или совместными усилиями группы пользователей, объединенных сходством решаемых задач. Эти средства предназначены специально для выбора компьютеров и ПО, наиболее подходящих под определенные прикладные задачи. В принципе такой подход позволяет получить наиболее точные оценки производительности для конкретного класса приложений, хотя и сопряжен со значительными усилиями пользователей по созданию тестовых программ и проведению испытаний компьютеров. Примером может служить известный пакет программ аэродинамических расчетов, применяемый для тестирования новых супер-ЭВМ в ИПМ им. М. В. Келдыша РАН.

Из числа стандартных тестов на сегодня наиболее распространенными являются наборы тестов компании SPEC (Standard Performance Evaluation Corporation) — SPEC (89, 92 и 95), тесты Linpack, тесты TPC-C (для OLTP-систем) и тесты NPB (для многопроцессорных ЭВМ).

Пакет тестовых программ **SPEC 89** включает два тестовых набора — Cint89, состоящий из четырех программ целочисленной обработки, и Cfp89, объединяющий шесть программ со значительным объемом операций над числами с плавающей точкой двойной точности. Все десять модулей представляют собой достаточно сложные программы на языках С и Fortran с широким спектром решаемых задач — от оптимизации матриц булевой логики до моделирования замещения атомов в квантовой химии. Методика оценки производительности SPEC 89 предполагает формирование десяти дифференциальных оценок SPECratioi, каждая из которых определяется, как отношение времени выполнения программы № i из наборов Cint89 и Cfp89 на тестируемом компьютере ко времени выполнения той же программы на ЭВМ фирмы DEC VAX 11/780.

Интегральной характеристикой производительности компьютера служит показатель SPECmark, являющийся средним геометрическим всех десяти частных оценок SPECratio. К параметру SPECmark добавлены еще две оценки — SPECint89 и SPECfp89, раздельно характеризующие быстродействие компьютера при обработке целочисленных данных и чисел с плавающей точкой соответственно.

Методика расчета основных характеристик производительности с помощью пакета **SPEC 92** по сравнению с пакетом SPEC 89 не претерпела никаких принципиальных изменений. Показатели SPECint92 и SPECfp92 по-прежнему определяются средними геометрическими частных оценок SPECratio, полученных при тестовом прогоне программ из наборов Cint92 и Cfp92 (конечно, с учетом увеличения числа тестовых программ).

В SPEC 92, по сравнению с пакетом SPEC 89, введено одно новшество. Речь идет о характеристиках мультипрограммной обработки SPECrate, формируемых в рамках метода однородной нагрузки. Суть последнего заключается в следующем: тестируемый компьютер выполняет задание, состоящее из множества копий одной программы, а показателем производительности многопроцессорной обработки служит количество копий, завершенных в определенный интервал времени. Для получения оценки SPECrate используются те же программы, что и для расчета показателей SPECint92 и SPECfp92. Разница только в том, что тестовый модуль реализуется как несколько копий, образующих одно задание, а результатом измерений является нормированное общее время выполнения всех копий задания. Подобной процедуре подвергается каждая из 20 тестовых программ, что позволяет получить шесть частных оценок SPECratio для программ целочисленной обработки и 14 — для программ обработки чисел с плавающей точкой.

Дальнейшее развитие микропроцессоров (повышение производительности и увеличение объема внутрикристальной памяти, совершенствование компиляторов и т. п.) привело к появлению нового тестового пакета **SPEC 95**.

В этом пакете индексы производительности даются по отношению к эталонной машине SPARC-station 10/40 в конфигурации с кэш-памятью второго уровня.

Используются два тестовых набора CINT 95 и CFP 95, состоящие из 8 и 10 программ соответственно.

При испытаниях компьютеров формируются:

- индексы производительности SPEC int 95, SPEC fp 95 и SPEC intbase 95, SPEC fpbase 95 для фиксированной и плавающей точки в оптимизированном режиме компиляции и без оптимизации соответственно;
- индексы пропускной способности SPEC intrate 95, SPEC fprate 95 и SPEC intratebase 95, SPEC fpratebase 95 для оценки многозадачных режимов и SMP-архитектур при оптимизированном режиме компиляции и без оптимизации соответственно.

Все интегральные индексы производительности формируются как среднее геометрическое индексов по отдельным тестам. При этом корпорация SPEC не

дает никаких рекомендаций по установлению соответствия между значениями индексов SPEC 92 и SPEC 95.

Набор тестов **Linpack** — это пакет программ для решения систем линейных алгебраических уравнений, разработанный известным американским ученым Джеком Донгаррой (Jack Dongarra). Алгоритмы линейной алгебры весьма широко используются в самых разных задачах, и поэтому измерение производительности на Linpack представляют интерес для многих пользователей.

В основе алгоритмов действующего варианта Linpack лежит метод декомпозиции. Исходная матрица размером 1000 × 1000 элементов сначала представляется в виде произведения двух матриц стандартной структуры, над которыми затем выполняется собственно алгоритм нахождения решения. Подпрограммы, входящие в Linpack, структурированы. В стандартном варианте Linpack выделен внутренний уровень базовых подпрограмм, каждая из которых выполняет элементарную операцию над векторами. Набор базовых подпрограмм называется BLAS (Basic Linear Algebra Subprograms). Например, в BLAS входят две простые подпрограммы SAXPY (умножение вектора на скаляр и сложение векторов) и SDOT (скалярное произведение векторов). Все операции выполняются над числами с плавающей точкой, представленными с двойной точностью. Результат измеряется в MFLOPS.

Использование результатов работы тестового пакета Linpack с двойной точностью, как основы для определения величины MFLOPS, стало общепринятой практикой в компьютерной промышленности. Для многопроцессорных систем также имеются параллельные версии Linpack, и такие системы часто показывают линейное увеличение производительности с ростом числа процессоров.

По результатам тестирования с использованием тестов Linpack с июня 1993 г. в университете Маннгейма (Mannheim) в Германии и в Netlib в США ведется так называемый список Top500, включающий 500 самых производительных компьютеров в мире. Список обновляется каждые полгода, попадание в этот список свидетельствует о значительном достижении той или иной страны или компании в области построения высокопроизводительных ЭВМ. Текущая версия списка Top500 доступна в Интернет по адресу http://www.top500.org.

По мере расширения использования компьютеров при обработке транзакций в сфере бизнеса все более важной становится возможность справедливого сравнения систем между собой. С этой целью в 1988 г. был создан Совет по оценке производительности обработки транзакций — TPC (Transaction Processing Performance Council). На сегодня членами TPC являются практически все крупнейшие производители аппаратных средств и программного обеспечения для автоматизации коммерческой деятельности.

В компьютерной индустрии термин «транзакция» (transaction) может означать почти любой вид взаимодействия или обмена информацией. Однако в мире бизнеса «транзакция» имеет вполне определенный смысл: коммерческий обмен

товарами, услугами или деньгами. Наиболее характерными примерами систем обработки транзакций являются системы управления учетом, системы резервирования авиабилетов и банковские системы.

Основной задачей организации TPC является точное определение тестовых пакетов для оценки систем обработки транзакций и баз данных, а также для распространения объективных, проверяемых данных в промышленности. Первоначально основными тестами для оценки производительности компании TPC являлись тесты TPC-A, TPC-B, TPC-C. В дальнейшем этот набор был расширен спецификациями тестов TPC-D и TPC-E. Организацией TPC ведутся постоянные исследования по модификации уже существующих и разработке новых спецификаций тестов.

Следует отметить, что, как и любой другой тест, ни один тест TPC не может измерить производительность системы, которая применима для всех возможных сред обработки транзакций, но эти тесты могут помочь пользователю сравнивать похожие системы.

Тест TPC-A, созданный в 1989 г., используется для оценки эффективности исполнения транзакций типа «дебит-кредит». Этот тест определяет пропускную способность системы, измеряемую количеством транзакций в секунду, которые система может выполнить при работе с множеством терминалов. Тест TPC-A может выполняться в локальных или региональных вычислительных сетях. При исполнении теста моделируется работа филиальной сети банка по приему-выдаче вкладов клиентов.

В августе 1990 г. был разработан тест TPC-B, интенсивный тест базы данных. Тест TPC-B также измеряет пропускную способность системы в транзакциях в секунду, но использует отличную от TPC-A методику расчетов (в частности, в TPC-B не выполняется эмуляция терминалов и линий связи).

Тестовый пакет TPC-C моделирует прикладную задачу обработки заказов. Он моделирует достаточно сложную систему OLTP, которая должна управлять приемом заказов, управлением учетом товаров и распространением товаров и услуг. Тест TPC-C осуществляет тестирование всех основных компонентов системы: терминалов, линий связи, процессоров, дискового ввода-вывода и базы данных. При исполнении теста моделируется функционирование нескольких складов компании. Результатом теста являются две величины. Одна из них, tpm-C, представляет пиковую скорость выполнения транзакций (выражается в количестве транзакций в минуту). Второй результат, $/tpm-C, представляет собой нормализованную стоимость системы. Стоимость системы включает все аппаратные средства и программное обеспечение, используемые в тесте, плюс стоимость обслуживания в течение 5 лет.

Тест TPC-D предназначен для оценки производительности систем принятия решений, а тест TPC-E — его альтернативный вариант для оценки систем масшта-

ба предприятия. Кроме того, TPC продолжает разработку тестовых пакетов для оценки баз данных и систем «клиент-сервер».

Тесты **NPB (NAS Parallel Benchmark)** разработаны Национальным космическим агентством (НАСА) США специально для измерения производительности многопроцессорных ЭВМ. Комплекс тестов NAS состоит из пяти тестов NAS kernel benchmark (оценки производительности ядра) и трех тестов, основанных на реальных задачах гидро- и аэродинамического моделирования. На сегодняшний день набор NPB является лучшим общепризнанным комплексом тестов для оценки параллельных многопроцессорных систем.

Комплекс тестов NAS Benchmarks kernel включает следующие расчетные задачи:

1) *EP (Embarrasinghly Parallel)* — вычисление интеграла методом Монте-Карло. Этот тест минимального межпроцессорного взаимодействия и определяет вычислительные характеристики узла многопроцессорной ВС при работе с вещественной арифметикой;

2) *MG (3D Multigrid)* — тест по решению уравнения Пуассона («трехмерная решетка») в частных производных;

3) *CG (Conjugate Gradient)* — вычисление наименьшего собственного значения больших, разреженных матриц методом сопряженных градиентов;

4) *FT (fast Fourier Tranformation)* — вычисление методом быстрого преобразования Фурье трехмерного уравнения в частных производных;

5) *IS (Integer Sort)* — тест выполняет сортировку целых чисел.

Комплекс тестов NAS Benchmarks по модельным задачам включает следующие модули:

1) *LU (LU Solver)* — тест выполняет вычисления, связанные с определенным классом алгоритмов (INS3D-LU по классификации центра NASA Ames), в которых решается система уравнений с равномерно разреженной блочной треугольной матрицей 5 × 5;

2) *SP (Scalar Pentadiagonal)* — тест выполняет решение нескольких независимых систем скалярных уравнений — пентадиагональные матрицы с преобладанием недиагональных членов;

3) *BT (Block Tridiagonal)* — решение серии независимых систем уравнений — блочные трехдиагональные матрицы 5 × 5 с преобладанием недиагональных элементов.

Основной вывод, который с очевидностью следует из вышесказанного о тестах оценки производительности, состоит в том, что результаты тестирования по разным тестам несопоставимы. Поэтому, если известна область применения ЭВМ или ВС, всегда следует отдавать предпочтение не оценке, полученной как среднее геометрическое оценок на разных тестах, а тестам, адекватным области применения.

1.5.4. Время ответа ЭВМ и ВС

Время ответа (Time of answer) — это длительность промежутка времени от момента поступления задания в систему до момента окончания его выполнения.

В общем случае время ответа — случайная величина, что обусловлено следующими факторами:

- влиянием исходных данных на число операций ввода, обработки и вывода данных и непредсказуемостью значений исходных данных;
- влиянием состава смеси задач, одновременно находящихся в системе, и непредсказуемостью состава смеси из-за случайности момента поступления задач на обработку.

Время ответа слагается из двух составляющих: *времени выполнения задачи* и *времени ожидания*. **Время выполнения задачи** при отсутствии параллельных процессов равно суммарной длительности всех этапов процесса — ввода, обращения к внешней памяти, процессорной обработки и вывода. Время выполнения задачи зависит от сложности вычислений и быстродействия устройств системы.

Время ожидания — сумма промежутков времени, в течение которых задача находилась в состоянии ожидания требуемых ресурсов. Ожидание возникает при мультипрограммной обработке, когда ресурс, необходимый задаче, занят другой задачей и первая задача не выполняется, ожидая освобождения ресурса. Время ожидания зависит главным образом от режима обработки задач и интенсивности входного потока заданий.

Среднее время ответа характеризует быстроту реакций системы на входные воздействия: задания, запросы абонентов и т. п. Качество системы тем выше, чем меньше среднее время ответа.

1.6. ПОКАЗАТЕЛИ НАДЕЖНОСТИ И ГОТОВНОСТИ ЭВМ И ВС

1.6.1. Основные понятия надежности ЭВМ и ВС

Важнейшей характеристикой вычислительных систем является надежность. Под **надежностью (reliability)** понимается свойство системы выполнять заданные функции, не изменяя во времени значения установленных эксплуатационных параметров, в заданных пределах, соответствующих определенным режимам и условиям эксплуатации, включающим условия использования, технического обслуживания, ремонта, хранения и транспортирования.

Понятие надежности тесно связано с другими показателями качества ЭВМ и ВС — *готовностью, отказоустойчивостью* и *точностью информации*.

В общем случае **готовностью** системы **(readiness)** называется способность технического устройства быть готовым к действию в любой момент времени. Применительно к ВС под этим показателем понимается свойство, характеризующее качество и быстроту выполнения операций по подготовке системы к применению. В число таких операций могут входить: включение элементов ВС, изменение не-

которых параметров системы, решение тестовых задач, изменение режима работы ВС и др.

В свою очередь, готовность ЭВМ и ВС складывается из следующих составляющих:

- доступность (availability);
- безотказность (reliability);
- безопасность (safety);
- ремонтопригодность (maintainability, repairs).

Доступность — это свойство системы находиться в состоянии готовности к работе. Обычно доступность показывает вероятность того, что система в данный момент времени будет правильно работать и окажется в состоянии выполнять свои функции. Другими словами, система с высокой степенью доступности — это такая система, которая в произвольный момент времени, скорее всего, находится в работоспособном состоянии.

Под **безотказностью** имеется в виду свойство системы работать без отказов в течение продолжительного времени. В противоположность доступности безотказность определяется в понятиях временного интервала, а не момента времени. Система с высокой безотказностью — это система, которая будет, скорее всего, непрерывно работать в течение относительно долгого времени. Между безотказностью и доступностью имеется небольшая, но существенная разница. Если система отказывает на одну миллисекунду каждый час, она имеет доступность порядка 99,9999%, но крайне низкую безотказность. Система же, которая никогда не отказывает, но раз в месяц отключается на два дня для профилактики, имеет высокую безотказность, но ее доступность составляет не более 94%.

Безопасность определяет, насколько критична ситуация временной неспособности системы должным образом выполнять заданные функции. Так, многие системы управления процессами, используемые, например, на атомных электростанциях или космических кораблях, должны обладать высокой степенью безопасности.

Ремонтопригодностью называется способность ВС к восстановлению в процессе эксплуатации. Данный показатель определяет, насколько сложно исправить возникающие неисправности в ЭВМ или ВС. Системы с высокой ремонтопригодностью могут также обладать высокой степенью доступности, особенно при наличии средств автоматического обнаружения и исправления неполадок.

Говорят, что система **отказывает (fail)**, если она не в состоянии выполнять свои функции. В частности, если система создавалась для предоставления пользователям некоторых услуг, то она будет считаться находящейся в состоянии отказа в том случае, если не сможет предоставлять все или некоторые услуги.

Ошибкой (error) называется такое состояние системы, которое может привести к ее неработоспособности. Так, например, при передаче пакетов по сети может случиться, что некоторые пакеты, пришедшие к получателю, окажутся

поврежденными. Повреждения в данном случае будут означать, что получатель может неверно прочесть значения битов (например, 0 вместо 1) или оказаться не в состоянии определить сам факт прихода пакета. Причиной ошибки является **отказ (fault)**. В общем случае, отказом системы называется такое ее поведение, которое не удовлетворяет ее спецификациям. Отказ системы может быть вызван отказом (неверным срабатыванием) каких-то ее компонентов (процессор, память, устройства ввода-вывода, линии связи или программное обеспечение).

Отказ компонента может быть вызван ошибками при конструировании, производстве или программировании. Он может быть также вызван физическим повреждением, изнашиванием оборудования, некорректными входными данными, ошибками оператора и многими другими причинами.

Отказы могут быть *случайными*, *периодическими* или *постоянными*.

Случайный отказ называется **сбоем**, при повторении операции случайные отказы исчезают. Причиной сбоя может служить, например, случайная электромагнитная помеха, резкий перепад напряжения питания и т. п.

Периодические отказы повторяются часто в течение какого-то времени, а затем могут долго не происходить. Классическим примером могут служить периодические сбои по причине плохого электрического контакта.

Постоянные (устойчивые) отказы не прекращаются до устранения их причины — разрушения диска, выхода из строя микросхемы или ошибки в программе.

По **характеру проявления** отказы подразделяются на *византийские* (система активна и может проявлять себя по-разному, даже злонамеренно) и *пропажа признаков жизни* (частичная или полная). Первые распознать гораздо сложнее, чем вторые. Свое название они получили по имени Византийской империи (330–1453), где процветали конспирация, интриги и обман.

По **степени влияния на работоспособность** ЭВМ или ВС различают *полные* (до устранения причин отказа система полностью неработоспособна) и *неполные* (неработоспособна только часть системы) отказы.

По **связи с другими отказами** выделяют *зависимые* и *независимые* отказы.

По **физическому характеру непосредственного проявления** отказы делятся на *катастрофические* и *параметрические*. **Катастрофический** отказ приводит к нарушению работоспособности всей системы, **параметрический** — к выходу за пределы нормы всех или части параметров системы.

Внезапным называется отказ, который возникает в результате скачкообразного изменения характеристик изделия. **Постепенный** отказ возникает в результате постепенного изменения характеристик изделия.

Причиной отказа служит **неисправность** ЭВМ или ВС. Неисправности подразделяют на *дефекты* и *неполадки*. **Дефектами** называются неисправности, которые в момент их обнаружения не приводят к повреждению или нарушению работы и регулировки системы, но могут в будущем вызвать подобные явления.

Неполадками называются неисправности в работе системы, оказывающие влияние на выполнение ею основных функций.

Для обеспечения надежного решения задач в условиях отказов ЭВМ или ВС применяются два принципиально различающихся подхода — **восстановление** после отказа ЭВМ или ВС и **предотвращение** отказа системы.

Повышение надежности основано на принципе предотвращения неисправностей путем снижения интенсивности отказов и сбоев за счет применения электронных схем и компонентов с высокой и сверхвысокой степенью интеграции, снижения уровня помех, облегченных режимов работы схем, обеспечения тепловых режимов их работы, а также за счет совершенствования методов сборки аппаратуры.

Надежность — это свойство ЭВМ и ВС, которое не надо путать с какими-либо количественными критериями, являющимися мерой той или иной стороны надежности. Следует помнить, что понятие надежности включает не только аппаратные средства, но и программное обеспечение. Главной целью повышения надежности систем является обеспечение целостности хранимых и обрабатываемых в ЭВМ и ВС данных.

Как уже отмечалось, помимо готовности, надежность тесно связана с отказоустойчивостью. **Отказоустойчивость** — это такое свойство ЭВМ и ВС, которое обеспечивает ей, как логической системе, возможность продолжения действий, заданных программой, после возникновения отказов. Обеспечение отказоустойчивости требует избыточного аппаратного и программного обеспечения. Направления, связанные с предотвращением неисправностей и с отказоустойчивостью, — основные в проблеме надежности. Концепции параллельности и отказоустойчивости вычислительных систем естественным образом связаны между собой, поскольку в обоих случаях требуются дополнительные функциональные компоненты. Поэтому, собственно, на параллельных вычислительных системах достигается как наиболее высокая производительность, так и во многих случаях очень высокая надежность. Имеющиеся ресурсы избыточности в параллельных системах могут гибко использоваться для повышения как производительности, так и надежности.

Точность информации относится к категории обобщенных показателей качества функционирования. Он носит системный характер, так как ошибки в выходной информации складываются из обычных ошибок, возникающих в ЭВМ при выполнении вычислений, и ошибок, появляющихся при передаче информации по каналам связи между машинами ВС, при сопряжении между собой машин с различными характеристиками (например, разрядность, форма представления чисел и др.). В случае если информация обрабатывается последовательным рядом машин, выходная ошибка предыдущей машины становится входной ошибкой последующей машины.

Следовательно, чтобы определить точность информации, надо хорошо знать характер ошибок, причины их возникновения и способы устранения. Информация, циркулирующая в ЭВМ или ВС, напрямую зависит от среды, где она может быть искажена. Здесь можно выделить три категории источников ошибок: оборудование, программное обеспечение и персонал.

В категории «*оборудование*» источником ошибок являются сами ЭВМ, линии связи, терминалы и другие вспомогательные средства.

В категории «*программное обеспечение*» источником ошибок являются трансляторы, редакторы, загрузчики, утилиты и другие программные средства.

В категории «*персонал*» источником ошибок являются стандарты документирования, квалификация разработчиков и эксплуатационников и др.

Наиболее типичными ошибками, влияющими на точность информации, являются:

- ошибки, связанные с неправильным вводом данных с устройства ввода (терминала, сканера и т. п.);
- ошибки, связанные с преобразованием информации из одной системы представления информации в другую;
- семантические ошибки при программировании;
- ошибки, связанные со сбоем оборудования и не приводящие к отказу всей ВС и др.

Очевидно, что чем выше надежность системы, тем более высокую точность информации она способна обеспечить.

Учитывая, что вопросы надежности ВС более подробно излагаются в соответствующем курсе, здесь будут рассмотрены показатели надежности и готовности в фрагментарном виде. Кроме того, будут разнесены вопросы, связанные с надежностью технических средств и программного обеспечения ВС.

1.6.2. Количественные показатели надежности, готовности и ремонтопригодности технических средств ЭВМ и ВС

Рассмотрим основные количественные характеристики надежности технических средств ЭВМ и ВС. Чаще всего в качестве таких характеристик используют:

1) вероятность безотказной работы системы за время t при заданных условиях эксплуатации — $P_c(t)$;
2) среднее время безотказной работы — T_c;
3) интенсивность отказов $\lambda(t)$ и время наработки на отказ T_o;
4) частоту отказов $\alpha(t)$.

Важнейшей характеристикой надежности является **интенсивность отказов**, определяющая среднее число отказов за единицу времени, как правило за один час. Интенсивность отказов зависит от числа элементов и соединений, составляющих систему.

Если любой отказ носит катастрофический характер, то интенсивность отказов в системе вычисляется по соотношению:

$$\lambda_0 = \sum_{i=1}^{n} \lambda_i, \tag{1.1}$$

где λ_i — интенсивность отказов i-го элемента или соединения элементов и n — число элементов или их соединений в системе. Так, если $\lambda_0 = 10^{-2}$, то в среднем за 100 ч происходит один отказ.

Средний промежуток времени между двумя отказами называется **средней наработкой на отказ** и равен

$$T_0 = \frac{1}{\lambda_0}. \tag{1.2}$$

Так, если $\lambda_0 = 10^{-2}$, то наработка на отказ составляет 100 ч. Промежутки времени между отказами — случайные величины со средним значением T_0, которые, как правило, распределены по экспоненциальному закону. При этом вероятность того, что за время, меньшее t, произойдет отказ (при экспоненциальном законе):

$$P(t) = 1 - e^{-\frac{t}{T_0}}. \tag{1.3}$$

Так, если $T_0 = 100$ ч, то вероятность того, что в течение 100 ч работы системы произойдет отказ, $P(t < 100) = 0{,}63$, и с вероятностью 0,37 отказ произойдет за время, больше 100 ч.

В то же время, зная интенсивность отказов, можно определить **вероятность безотказной работы** системы $P_c(t)$:

$$P_c(t) = e^{-\lambda_0 \cdot t}. \tag{1.4}$$

Среднее время безотказной работы T_c связано с вероятностью безотказной работы системы $P_c(t)$ за время t следующим соотношением:

$$T_c = \int_0^{\infty} P_c(t)dt. \tag{1.5}$$

Частота отказов $\alpha(t)$ — это плотность вероятностей времени работы ЭВМ или ВС с момента включения до ее отказа. Частота отказов наиболее полно характеризует надежность системы. Зная частоту отказов, можно определить остальные количественные показатели надежности.

Связь вероятности безотказной работы ВС $P_c(t)$ с частотой отказов $\alpha(t)$ определяется следующей формулой:

$$P_c(t) = 1 - \int_0^{t} \alpha(x)dx. \tag{1.6}$$

Работоспособность системы, нарушенная в результате отказа, восстанавливается путем ремонта системы. Ремонт состоит в выявлении причины нарушения работоспособности (*диагностике системы*) и в восстановлении работоспособности путем замены неисправного элемента. Основными количественными характеристиками ремонтопригодности ЭВМ и ВС являются:

1) вероятность восстановления системы за заданное время $P_{ВС}(t)$;

2) среднее время восстановления — $T_{ВС}$;

3) интенсивность восстановления $\mu(t)$.

Вероятность $P_{ВС}(t)$ является интервальным показателем, $T_{ВС}$ — интегральным, а $\mu(t)$ — точечным показателем ремонтопригодности.

Временем восстановления $T_{ВС}$ называется промежуток времени, затрачиваемый на восстановление работоспособности системы. Его величина зависит от сложности системы, степени совершенства средств диагностики и уровня ремонтопригодности системы, а также уровня подготовки обслуживающего персонала.

Интенсивность восстановления $\mu(t)$ — это условная плотность вероятности восстановления работоспособного состояния ЭВМ или ВС, определенная для рассматриваемого момента времени при условии, что до этого момента восстановление не было завершено.

В случае, когда интенсивность восстановления постоянна, т. е. $\mu(t) = \mu = \text{const}$, вероятность восстановления за заданное время t подчиняется экспоненциальному закону и определяется по выражению

$$P_{ВС}(t) = 1 - e^{-\mu \cdot t}. \tag{1.7}$$

Этот частный случай имеет наибольшее практическое значение, поскольку реальный закон распределения времени восстановления большинства ЭВМ и ВС (поток восстановлений) близок к экспоненциальному. В этом случае связь между интенсивностью и средним временем восстановления следующая:

$$T_{ВС} = \frac{1}{\mu}. \tag{1.8}$$

Готовность ЭВМ и ВС зависит от надежности и ремонтопригодности. Чем выше надежность и ремонтопригодность, тем выше готовность. Количественными показателями готовности являются **функция готовности $K_Г(t)$** и **коэффициент готовности $K_Г$**. Функция готовности есть вероятность того, что в любой момент времени t система готова к действию. Для технических средств ЭВМ и ВС характеристика обычно имеет вид, показанный на рис. 1.4.

Из рисунка видно, что $K_Г(0) = 1$, т. е. считается, что ЭВМ или ВС начинает эксплуатироваться исправной. С ростом t $K_Г(t)$ убывает и при $t \to \infty$ функция готовности стремится к постоянной, отличной от нуля величине, которая является финальной вероятностью и называется коэффициентом готовности. Таким образом, между функцией и коэффициентом готовности существует зависимость:

$$K_{\Gamma} = \lim_{t \to \infty} K_{\Gamma}(t). \tag{1.9}$$

Функция готовности является точечной характеристикой. Это означает, что координата $K_{\Gamma}(t)$, показанная на рис. 1.4, есть вероятность того, что в момент времени t система исправна. До момента t она могла сколько угодно раз отказывать и ремонтироваться.

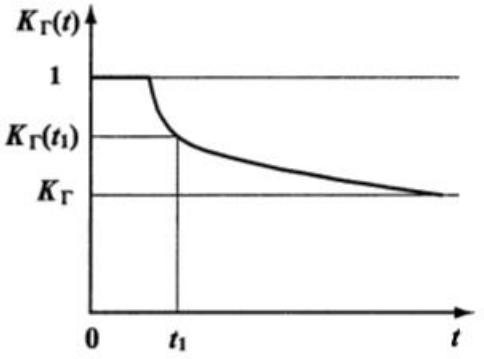


Рис. 1.4. График функции готовности технических средств ЭВМ и ВС

Коэффициент готовности является интегральной характеристикой и легко вычисляется, если известны интегральные характеристики надежности и ремонтопригодности:

$$K_{\Gamma} = \frac{T_0}{T_0 + T_{BC}}, \tag{1.10}$$

где T_0 — время наработки на отказ, $T_{\text{ВС}}$ — среднее время восстановления ВС.

1.6.3. Показатели надежности программных средств ЭВМ и ВС

При рассмотрении показателей надежности программных средств ЭВМ и ВС необходимо учитывать тот факт, что ПО изменяется не по мере наработки, а в результате устранения ошибок, т. е. вероятность отказа в работе не имеет прямого отношения к времени наработки. Поэтому при расчете количественных показателей надежности ПО необходимо в первую очередь учитывать свойства программ, входящих в его состав.

Основными исходными предпосылками при расчете показателей надежности ПО можно считать следующие:

1. Каждая программа, обрабатывающая множество N_i входных данных, дает однозначный выходной результат.

2. Множество N_i всех входных данных определяет все вычисления, выполняемые программой.

3. Каждая ошибка в программе вызывает сбои для некоторой части входных данных.

4. Выполнение программы с некоторым подмножеством входных данных представляет собой единичное наблюдение ее действия.

Вид *функции готовности* $K_г(t)$ для программных средств ЭВМ и ВС (рис. 1.5) отличается от вида аналогичной функции для технических средств. В начальный момент времени функция готовности имеет некоторое значение, отличное от 0 и 1, — в новом ПО имеются необнаруженные ошибки. В процессе эксплуатации ПО ошибки исправляются, и значение функции готовности растет со временем, приближаясь к 1. Однако процесс исправления «старых» ошибок неизбежно влечет за собой внесение «новых». В некоторый момент времени t_0 функция готовности достигает своего максимума, после чего, за счет вносимых «новых» ошибок, начинает убывать. В момент времени t_1 модификация ПО прекращается, и функция готовности становится постоянной.

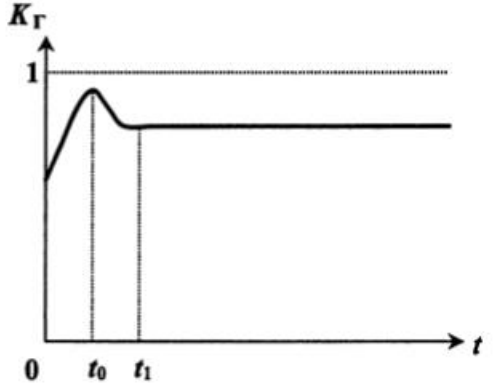


Рис. 1.5. График функции готовности программных средств ЭВМ и ВС

1.6.4. Методы и средства повышения надежности ЭВМ и ВС

Как и в отношении показателей надежности, будем рассматривать методы и средства повышения надежности раздельно для технических средств и ПО, так как наряду с общими принципами они имеют существенные различия.

Все методы повышения надежности для технических средств в зависимости от области их применения обычно делят на три группы: *производственные*, *схемно-конструкторские* и *эксплуатационные*.

Производственными считаются методы, определяющие пути повышения надежности в процессе создания элементов ЭВМ и ВС. К ним обычно относят:

1) получение однородной продукции;
2) стабилизацию технологии;
3) анализ дефектов и механизмов отказов;
4) исключение известных видов отказов;

5) разработку методов испытаний. Определение зависимостей показателей надежности от интенсивности внешних воздействий;

6) проведение ускоренных испытаний и тренировки изделий;

7) повышение культуры производства;

8) контроль качества изделий на всех участках технологического процесса.

Схемно-конструкторские методы повышения надежности используются инженерами-разработчиками на стадии проектирования ЭВМ и ВС. К ним относятся:

1) выбор подходящих уровней нагрузки;

2) унификацию элементов и узлов. Входной контроль элементов и узлов;

3) разработку схем с широкими допусками на отклонение параметров элементов;

4) резервирование;

5) контроль работы оборудования и введение избыточности по времени;

6) использование корректирующих кодов.

Эксплуатационные методы обеспечивают повышение надежности за счет организации технического обслуживания ЭВМ и ВС. Эта группа методов включает:

1) сбор информации по надежности ЭВМ и ВС;

2) коррекцию рабочих режимов ЭВМ и ВС;

3) проведение профилактических мероприятий;

4) обучение обслуживающего персонала.

Повышение надежности ПО связано со значительно большими трудностями, чем для технических средств. Это обусловлено следующими основными факторами:

- большое количество ошибок, сопровождающих разработку ПО;
- трудности в создании приемлемого для пользователя интерфейса;
- сложность самого ПО;
- необходимость наличия высококвалифицированного персонала, участвующего в разработке и эксплуатации ПО;
- трудности, связанные с разработкой качественной документации, сопровождающей ПО.

Эти трудности приводят к тому, что к настоящему времени пока не существует эффективной системы контроля качества программ.

Все факторы, определяющие надежность ПО, можно так же, как и для технических средств, разделить на три группы: факторы связанные с разработкой ПО (организационные и технологические), конструктивные и эксплуатационные факторы.

Факторы первой группы непосредственно влияют на обеспечение заданного уровня надежности. При этом конструктивные факторы напрямую зависят от накопленного опыта разработчиков и характеризуют качественную сторону надеж-

ности. Технологические факторы проявляются непосредственно при разработке ПО и зависят в первую очередь от используемой технологии программирования. Организационные факторы определяют надежность как следствие организации трудового коллектива. Эксплуатационные факторы определяют надежность с качественной стороны и не могут практически быть измерены количественно.

Следует заметить, что при реализации факторов из любой группы важным является уровень подготовки персонала, участвующего в разработке и эксплуатации ПО.

1.7. АДАПТИВНОСТЬ ЭВМ И ВС

Под **адаптивностью** системы понимается способность ВС к самоорганизации. ВС называется самоорганизующейся, если в ней на основании оценки воздействия внешней среды, путем последовательного изменения своих свойств, заложены возможности перехода к некоторому устойчивому состоянию, когда воздействия внешней среды оказываются в допустимых пределах.

С адаптивностью связан такой немаловажный показатель качества, как **масштабируемость**, под которой понимается возможность наращивания числа и мощности процессоров, объемов оперативной и внешней памяти и других ресурсов вычислительной системы. Масштабируемость должна обеспечиваться архитектурой и конструкцией ЭВМ или ВС, а также соответствующими средствами программного обеспечения. Другими словами, масштабируемость — это способность ЭВМ или ВС адаптироваться к изменению мощности вычислительных ресурсов.

Добавление каждого нового процессора или другого ресурса в действительно масштабируемой системе должно давать адекватное произведенным затратам увеличение производительности и пропускной способности. Одной из основных задач при построении масштабируемых систем является минимизация стоимости расширения системы и упрощение планирования. В идеале добавление вычислительных ресурсов к системе должно приводить к линейному росту ее производительности. Однако это не всегда так. Потери производительности могут возникать, например, при недостаточной пропускной способности шин из-за возрастания трафика между процессорами и основной памятью, а также между памятью и устройствами ввода-вывода.

Возможность масштабирования системы определяется не только архитектурой аппаратных средств, но зависит от заложенных свойств программного обеспечения. Масштабируемость программного обеспечения затрагивает все его уровни от простых механизмов передачи сообщений до работы с такими сложными объектами, как мониторы транзакций и вся среда прикладной системы. В частности, программное обеспечение должно минимизировать трафик межпроцессорного обмена, который может препятствовать линейному росту производительности системы. Аппаратные средства (процессоры, шины и устройства ввода-вывода)

являются только частью масштабируемой архитектуры, на которой программное обеспечение может обеспечить предсказуемый рост производительности. Важно понимать, что простой переход, например, на более мощный процессор может привести к перегрузке других компонентов системы. Это означает, что действительно масштабируемая система должна быть сбалансирована по всем параметрам.

Следующим показателем качества, связанным со способностью адаптации ЭВМ или ВС, является **совместимость и мобильность ПО**.

Концепция программной совместимости впервые в широких масштабах была применена разработчиками системы IBM S/360. Основная задача при проектировании всего ряда моделей этой системы заключалась в создании такой архитектуры, которая была бы одинаковой с точки зрения пользователя для всех моделей системы независимо от цены и производительности каждой из них. Огромные преимущества такого подхода, позволяющего сохранять существующий задел программного обеспечения при переходе на новые (как правило, более производительные) модели были быстро оценены как производителями компьютеров, так и пользователями. Начиная с этого времени, практически все фирмы — поставщики компьютерного оборудования взяли на вооружение эти принципы, поставляя серии совместимых компьютеров. Однако следует заметить, что со временем даже самая передовая архитектура неизбежно устаревает и возникает потребность внесения радикальных изменений в архитектуру и способы организации вычислительных систем.

В настоящее время одним из наиболее важных факторов, определяющих современные тенденции в развитии информационных технологий, является ориентация компаний — поставщиков компьютерного оборудования на рынок прикладных программных средств. Это объясняется прежде всего тем, что для конечного пользователя в конце концов важно программное обеспечение, позволяющее решить его задачи, а не выбор той или иной аппаратной платформы.

Переход от однородных сетей программно совместимых компьютеров к построению неоднородных сетей, включающих компьютеры разных фирм-производителей, в корне изменил и точку зрения на саму сеть: из сравнительно простого средства обмена информацией она превратилась в средство интеграции отдельных ресурсов — мощную распределенную вычислительную систему, каждый элемент которой (сервер или рабочая станция) лучше всего соответствует требованиям конкретной прикладной задачи.

Этот переход выдвинул ряд новых требований. Во-первых, такая вычислительная среда должна позволять гибко менять количество и состав аппаратных средств и программного обеспечения в соответствии с меняющимися требованиями решаемых задач. Во-вторых, она должна обеспечивать возможность запуска одних и тех же программных систем на различных аппаратных платформах, т. е. обеспечивать *мобильность программного обеспечения*. В третьих, эта среда долж-

на гарантировать возможность применения одних и тех же человеко-машинных интерфейсов на всех компьютерах, входящих в неоднородную сеть. В условиях жесткой конкуренции производителей аппаратных платформ и программного обеспечения сформировалась *концепция открытых систем*, представляющая собой совокупность стандартов на различные компоненты вычислительной среды, предназначенных для обеспечения мобильности программных средств в рамках неоднородной, распределенной вычислительной системы.

Одним из вариантов моделей открытой среды является модель OSE (Open System Environment), предложенная комитетом IEEE POSIX. На основе этой модели национальный институт стандартов и технологии США выпустил документ «Application Portability Profile (APP). The U.S. Government's Open System Environment Profile OSE/1 Version 2.0», который определяет рекомендуемые для федеральных учреждений США спецификации в области информационных технологий, обеспечивающие мобильность системного и прикладного программного обеспечения. Все ведущие производители компьютеров и программного обеспечения в США в настоящее время придерживаются требований этого документа.

1.8. ЭКОНОМИЧНОСТЬ ЭВМ И ВС

Экономичность ЭВМ или **ВС** оценивается затратами C на разработку и эксплуатацию:

$$C = C_{\text{эвм}} + C_{\text{экспл}}, \tag{1.11}$$

где $C_{\text{эвм}}$ — стоимость самой ЭВМ или ВС; $C_{\text{экспл}}$ — стоимость затрат на эксплуатацию ЭВМ или ВС. В стоимость входит стоимость как технических средств, так и программного обеспечения.

Стоимость технических средств является установившейся категорией. Существует целый ряд методик, позволяющих достаточно точно оценить затраты на создание ЭВМ и ВС.

При оценке стоимости ПО в общем случае нужно различать стоимость разработки и стоимость его сопровождения. На стоимость разработки ПО влияют следующие три группы факторов: 1) факторы, связанные с формулированием задачи проектирования; 2) факторы, связанные с организацией проектирования; 3) факторы, определяемые влиянием окружающей среды. Перечень этих факторов приведен на рис. 1.6.

В первой группе наиболее емким по затратам является фактор, связанный с объемом исходного текста ПО. Для облегчения расчетов стоимости стремятся получить линейную зависимость одной разработанной команды и ее стоимости за счет коррекции влияния других факторов этой группы. При этом стоимость одной команды определяется исходя из оплаты труда программиста. Так, например, если определить годовую оплату среднего программиста в 35 тыс. долларов (высококвалифицированный программист имеет годовую оплату до 60 тыс. долларов) и его производительность в (15–30) команд в день, то стоимость одной команды на

языке ассемблера составит от 12 до 24 долларов, а на языке высокого уровня — от 6 до 12 долларов.

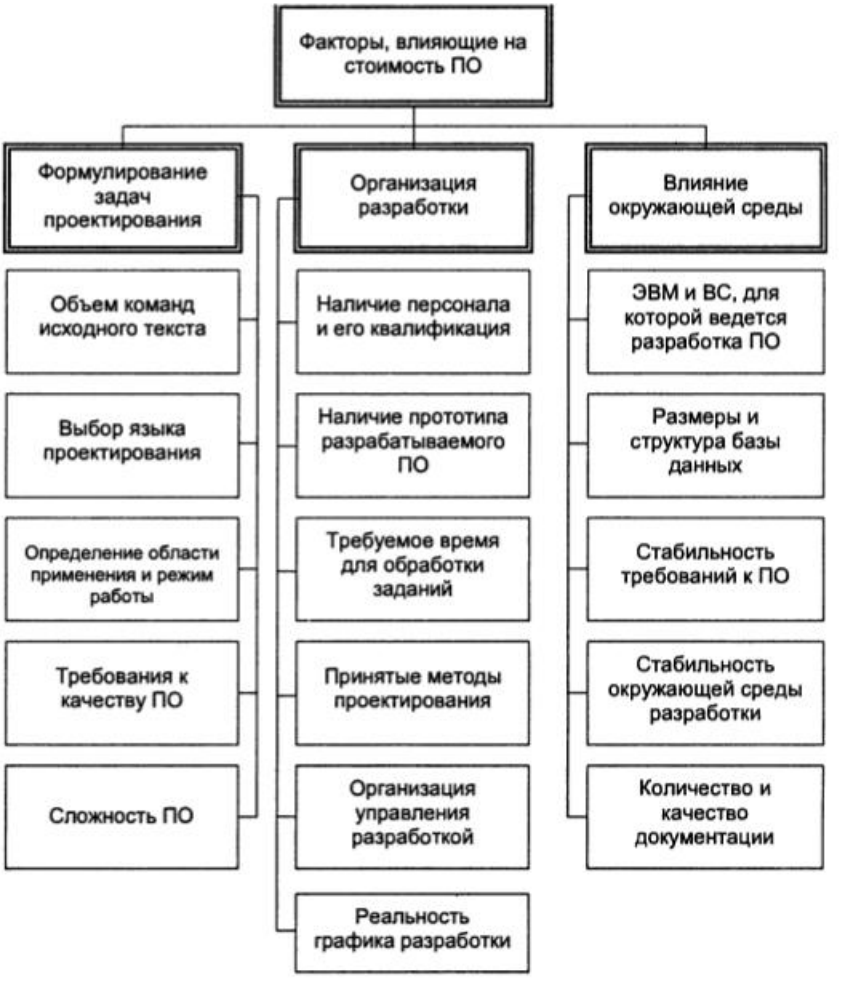


Рис. 1.6. Перечень факторов, определяющих стоимость ПО

Важную роль в определении стоимости ПО играет область применения ПО. Естественно, что стоимость ПО, работающего в реальном времени, будет больше стоимости ПО, работающего в других режимах. Исходя из оценки уже разработанных систем, можно сказать, что стоимость ПО реального времени примерно в 5 раз больше, чем стоимость ПО для других приложений. Затраты на качество ПО определяются в основном затратами на тестирование. Влияние сложности на стоимость ПО — субъективный фактор, и его оценки являются весьма приближенными.

Во второй группе факторов наибольшее влияние на стоимость ПО имеет персонал, участвующий в разработке. В первую очередь это связано с квалификацией, а, следовательно, и с производительностью труда.

Для факторов третьей группы наибольшее значение имеет ЭВМ и ВС, для которой ведется разработка ПО. Важным здесь в первую очередь является архитектура, производительность, объем памяти и другие характеристики. Не менее важным является количество и качество документации на ПО. Если документация на ПО готовится вручную, то стоимость одной страницы текста колеблется в пределах от 35 до 100 долларов, с учетом автоматизации эти цены уменьшаются на порядок.

Все приведенные факторы позволяют дать качественную оценку затрат на проектирование ПО. Сложнее дать количественную оценку затрат. Единой методики в этом вопросе нет, а существует ряд подходов. Рассмотрим некоторые из них.

Оценивание сверху вниз.

При этом подходе сметчик (лицо, выполняющее оценку) основывается на общей стоимости или стоимости отдельных крупных составных частей проекта. Пользуясь накопленным опытом и интуицией, он определяет стоимости отдельных компонент проекта. Как видно, этот подход сугубо субъективистский и определяется только квалификацией сметчика.

Оценка соотношений.

Иногда этот подход называют сопоставлением по эталону. При этом подходе необходимо иметь в базе данных некоторые оценки затрат на создание того или иного компонента ПО. Основываясь на коэффициентах соотношения эталона и оцениваемого объекта, вырабатывается оценка проектируемого компонента ПО. Предварительно каждый модуль оценивается по возможному числу команд объектного кода, классифицируется по их типу и сложности. На основе данных стоимостей из базы данных в терминах стоимости одной команды и уровня сложности ПО строится матрица стоимости.

Оценивание по стандартам.

Этот метод базируется на известных (стандартных) нормах затрат на выполнение единицы работы. Метод прост, доступен и точен только тогда, когда одинаковые операции выполняются повторно и имеются хорошо отработанные документы по объектам оценки.

Оценка снизу вверх.

При использовании этой методики вся оцениваемая работа делится на небольшие части, или единицы работы, имеющие самостоятельный характер. Как правило, такая часть должна иметь характер самостоятельной (элементарной) задачи. После этого каждая элементарная часть оценивается с точки зрения стоимости. Суммированием затрат на все единицы работ получается итоговая стоимость

всего проекта. Преимущество данной методики заключается в ее сравнительной простоте, что снижает требования к квалификации сметчика.

Следует заметить, что затраты на сопровождение уже разработанного ПО могут оказаться значительно большими стоимости самого ПО. По данным некоторых американских специалистов следует, что если затраты на разработку одной команды ПО составляют до 75 долларов, то на сопровождение — могут составлять до 4000 долларов на команду.

Зачастую приходится заниматься модификацией как всей ВС, так и отдельно ее технических или программных средств. Поэтому важным является вопрос об оценке стоимости таких работ. Будем понимать под модификацией ВС такое ее изменение, которое приспосабливает ее к выполнению целевых функций в прежних условиях эксплуатации.

Как правило, стоимость ВС, которую необходимо модифицировать, известна, также известна и величина ее износа (это касается технической составляющей ВС). Следует заметить, что наиболее значительно снижаются затраты на модификацию технических средств по сравнению с созданием новых средств. Снижение затрат достигается за счет их уменьшения на проектирование, испытания, серийное производство и эксплуатацию.

Вопросы для самоконтроля

1. Дайте определения основным терминам в области ЭВМ, ВС и комплексов.
2. Что понимается под архитектурой ЭВМ и ВС?
3. Какие существуют основные классы ЭВМ? По каким признакам можно классифицировать ВС?
4. В чем состоит различие персональной ЭВМ и рабочей станции?
5. Какие существуют классы персональных ЭВМ?
6. Приведите основные характеристики и классификацию ноутбуков.
7. Каковы отличительные черты карманных ПК?
8. Охарактеризуйте назначение и основные тенденции развития серверов.
9. Каковы основные преимущества и недостатки мэйнфреймов?
10. Чем отличается мэйнфрейм от супер-ЭВМ?
11. Что такое «показатель качества ЭВМ и ВС»? Какие бывают показатели качества?
12. В чем заключается эффективность ЭВМ и ВС?
13. Что такое пиковая производительность ЭВМ или ВС, и в чем она измеряется?
14. Как определить реальную производительность ЭВМ или ВС?
15. Из чего складывается время ответа ЭВМ или ВС?
16. Как связаны надежность, готовность, отказоустойчивость и точность информации ЭВМ или ВС?

17. Назовите виды отказов ЭВМ и ВС.

18. Охарактеризуйте основные количественные показатели надежности и ремонтопригодности технических средств ЭВМ и ВС.

19. Какой вид имеет функция готовности технических средств ЭВМ и ВС и почему?

20. Какой вид имеет функция готовности программных средств ЭВМ и ВС и почему?

21. Перечислите методы и средства повышения надежности ЭВМ и ВС.

22. Что такое адаптивность и масштабируемость ЭВМ и ВС?

23. В чем заключается совместимость и мобильность ПО?

24. Из чего складываются затраты на ЭВМ и ВС?

25. Назовите факторы, определяющие стоимость ПО.

26. Какие подходы применяются при оценке стоимости ПО?

ГЛАВА 2. ЭВМ ПОСЛЕДОВАТЕЛЬНОГО ТИПА

2.1. СТРУКТУРА, ОСНОВНЫЕ КОМПОНЕНТЫ И ПРИНЦИПЫ ФУНКЦИОНИРОВАНИЯ ПОСЛЕДОВАТЕЛЬНЫХ ЭВМ

Под **ЭВМ последовательного типа** будем понимать вычислительную машину, построенную в соответствии с принципами, выдвинутыми в 1945 г. выдающимся американским ученым венгерского происхождения **Джоном фон Нейманом**. Принципы фон Неймана до сих пор определяют так называемую *традиционную (классическую)* или *фон-неймановскую* архитектуру ЭВМ.

Фон-неймановская последовательная ЭВМ (рис. 2.1) состоит из

- **центрального процессора** (ЦП), включающего **арифметико-логическое устройство** (АЛУ) и **устройство управления** (УУ);
- **оперативной памяти** (ОП);
- **каналов (портов) ввода-вывода**, к которым подключаются разнообразные **устройства ввода-вывода** (УВВ), в том числе **внешние запоминающие устройства** или **внешняя память** (ВЗУ или ВП).

ЦП, кроме АЛУ и УУ, может иметь собственную внутреннюю память (блок управляющих регистров; блок регистровой памяти и др.). Сразу отметим, что важнейшим управляющим регистром ЦП является **программный счетчик** (PC — Program Counter, другое название — **счетчик команд, или указатель команды**, IP, Instruction Pointer), в котором хранится адрес следующей выполняемой команды. Информация, хранимая в ОП, непосредственно доступна центральному процессору, т. е. ЦП может произвольно читать и изменять любой участок ОП. В отличие от ОП информация, хранимая во внешней памяти, доступна центральному процессору только через каналы ввода-вывода.

Часть технических средств, включающая объединенные единым управлением ЦП, ОП и КВВ, называется **центральной частью** ЭВМ. УВВ включают набор различных устройств для ввода и вывода информации, совокупность УВВ называют **периферийной частью** ЭВМ.

Рассмотрим принципы фон Неймана, в соответствии с которыми происходит функционирование классической последовательной ЭВМ.

1. **Принцип двоичного кодирования**. Согласно этому принципу компьютеры на электронных элементах должны работать в двоичной системе счисления. Это означает, что любая информация внутри ЭВМ записывается и хранится в двоичной форме. Напомним, что минимальной единицей информацией в двоичной системе счисления является двоичный разряд или **бит**, принимающий значения 0 или 1. Биты обычно объединяются в группы по 8 разрядов, называемые **байтами**. Любая информация, поступающая в ЭВМ, кодируется с помощью двоичных сигналов и переводится в битовое или байтовое представление.

2. **Принцип адресности**. Структурно оперативная память состоит из пронумерованных ячеек, номер ячейки называется **адресом памяти**. Центральному процессору в произвольный момент времени по адресу доступна любая ячейка. Отсюда следует возможность давать имена областям памяти, так чтобы к запомненным в них значениям можно было бы впоследствии обращаться или менять их в процессе выполнения программы с использованием присвоенных имен.

3. **Принцип однородности памяти**. В оперативной памяти ЭВМ хранятся и программы, и данные. При этом сущностного различия между программой и данными нет. Центральный процессор может считывать и записывать любую ячейку (область) памяти независимо от того, хранится в ней часть программы или данные. Более того, *наличие программы в ОП является необходимым условием ее выполнения*. Помещение программы в ОП осуществляется центральным процессором путем считывания ее кода из ВЗУ и записи в ОП. Из принципа однородности памяти следует возможность модификации кода программы во время ее выполнения.

4. **Принцип программного управления**. Из него следует, что программа состоит из упорядоченного набора **команд**, которые выполняются центральным процессором автоматически друг за другом в определенной последовательности. Сама исполняемая программа хранится в ОП, причем выборка из ОП очередной команды для исполнения осуществляется центральным процессором согласно значению программного счетчика. После выборки из очередной команды значение программного счетчика автоматически увеличивается на длину выбранной команды. Программист может изменять значение программного счетчика командами условных и безусловных переходов, вызова и возврата из подпрограммы, вызова прерывания (подробнее см. п. 2.3.3).

Рассмотрим назначение основных компонентов ЭВМ последовательного типа (рис. 2.1).

Арифметико-логическое устройство (АЛУ) производит арифметические и логические преобразования над поступающими в него машинными словами, т. е. кодами определенной длины, представляющими собой числа или другой вид информации.

По способу обработки операндов АЛУ делятся на *последовательные* и *параллельные*. В **последовательных** АЛУ операнды представляются в последовательном коде и операции выполняются последовательно над каждым разрядом. В **параллельных** АЛУ операнды представляются в параллельном коде и операции выполняются параллельно над всеми разрядами операнда.

По характеру использования элементов различают *блочные* и *многофункциональные* АЛУ. В **блочном** АЛУ операции над различными данными выполняются в отдельных блоках (например, отдельный блок для операций над целыми числами и отдельный блок — для операций над числами с плавающей запятой). Это повышает скорость обработки, но увеличиваются затраты на оборудование.

В **многофункциональных** АЛУ операции над всеми типами данных выполняются одними и теми же схемами. В этом случае скорость обработки меньше, но и затраты на оборудование сокращаются.

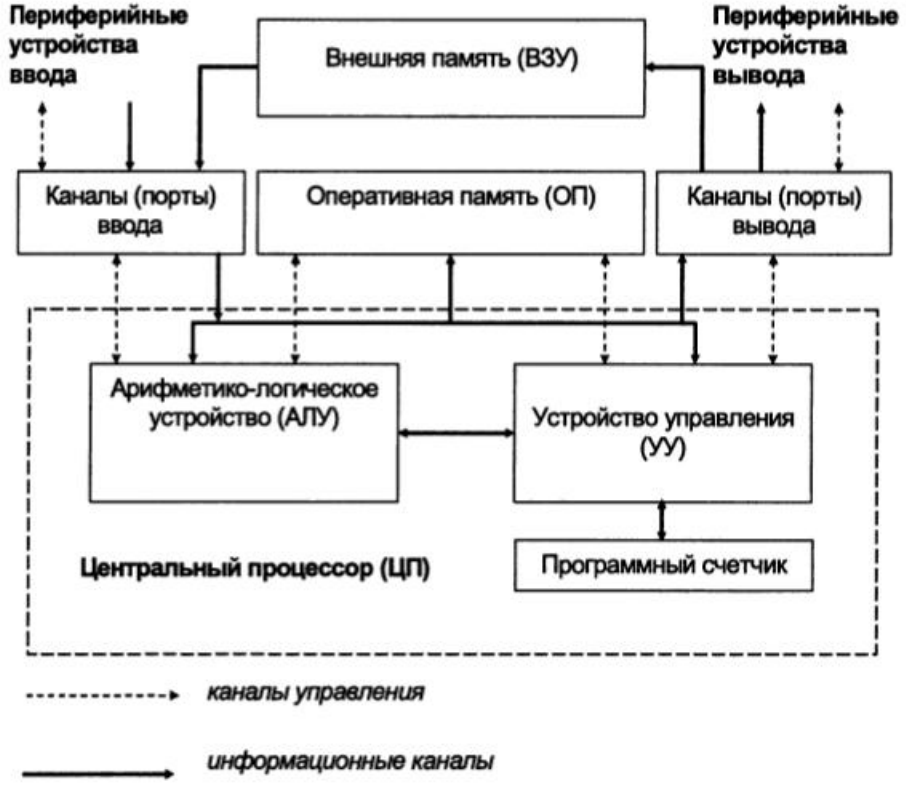


Рис. 2.1. Структура ЭВМ последовательного типа

Устройство управления (УУ) автоматически управляет вычислительным процессом, посылая всем другим устройствам сигналы, предписывающие им те или иные действия. В частности, УУ указывает ОП, какие слова должны быть переданы в АЛУ и в другие устройства, включает АЛУ на выполнение нужной операции и помещает полученный результат в ОП.

Память хранит информацию, передаваемую из других устройств, в том числе поступающую в машину извне через устройство ввода, и выдает во все другие устройства информацию, необходимую для осуществления вычислительного процесса.

Устройство ввода предназначено для ввода информации в ЭВМ. Ввод информации осуществляется либо с клавиатуры, либо с помощью сканера, либо речевым или другим способом.

Устройство вывода служит для выдачи из машины результатов расчета, например, путем печатания их на печатающих устройствах (ПУ), отображения на экране дисплея или выдачи в канал связи.

Как уже упоминалось, программа для фон-неймановской машины состоит из размещенных в ОП набора команд, данных и программного счетчика, определяющего в каждый момент времени команду, которая будет выполнена следующей. При этом, какова бы ни была архитектура системы команд ЭВМ, каждая команда содержит, как минимум, три обязательных поля:

- код операции;
- способ адресации операндов;
- адреса операндов.

Кодом операции определяется, *какое действие* будет выполнено центральным процессором по этой команде. Например, сложение двух чисел, которые в этом случае будут выступать в качестве *операндов*. Операнды суть входные данные для команды. В фон-неймановской машине программист должен вычислить (подготовить) операнды заранее, до того как использующая эти операнды команда поступит на выполнение. Местонахождение операндов (исполнительный адрес) определяется в соответствии с заданным в команде *способом адресации*.

Рассматривая структуру последовательных ЭВМ, следует заметить, что возможны два подхода к организации управления выполнением команд в центральном процессоре: *аппаратное* и *микропрограммное*. **Аппаратное управление** предполагает установление соответствия между кодом операции и последовательностью вырабатываемых управляющих сигналов с помощью фиксированных соединений между блоками процессора. Такой подход характеризуется высокой топологической сложностью управляющей подсистемы процессора.

Микропрограммное управление предполагает организацию распределенного во времени управления в соответствии с последовательностью микрокоманд (микропрограмм), определяемой по коду операции выполняемой команды процессора. Микропрограммы хранятся в постоянном запоминающем устройстве, которое называют *постоянным запоминающим устройством микропрограмм*. Микропрограммное управление характеризуется простотой реализации и возможностями расширения системы команд процессора.

Можно констатировать, что процесс функционирования во времени устройства обработки цифровой информации состоит из последовательности типовых интервалов, в течение которых выполняются элементарные операции (передача слова, сдвиг и др.).

Такая элементарная операция, выполняемая за один тактовый интервал, называется *микрооперацией (микрокомандой)*. Последовательность таких микрокоманд, реализующих данную операцию (команду), называется *микропрограммой*. Для определения временных соотношений между различными этапами операции используется понятие *машинного такта*, т. е. интервала времени, в течение ко-

торого выполняется одна или одновременно несколько микроопераций. Границы тактов задаются синхросигналами.

Развитие принципов фон Неймана в конкретных реализациях направлено на повышение производительности, эффективности, надежности и других показателей качества вычислительной системы. Например, повысить эффективность использования вычислительных ресурсов системы удалось с помощью введения мультипрограммирования. В рамках фон-неймановской модели это означало создание и поддержание множества копий программных счетчиков. Мультипрограммирование может быть реализовано как в однопроцессорной, так и в мультипроцессорной системах. Каждый из программных счетчиков связан со своей программой, а изменения значений счетчиков могут происходить упорядоченно (в однопроцессорной системе) или асинхронно (в мультипроцессорном варианте). Другой пример. Уменьшить время обращения к памяти системы удалось, обеспечив возможность одновременной выборки данных и команд благодаря логическому и физическому разнесению памяти команд и данных. Такая архитектура памяти получила название *гарвардской*, так как впервые была реализована в 1944 г. в Гарвардском университете при создании релейной ЭВМ.

2.2. ЦИКЛ КОМАНДЫ

Как уже упоминалось, функционирование фон-неймановской ЭВМ представляет собой последовательное выполнение центральным процессором команд, образующих программу. Все команды программы размещаются в оперативной памяти. Действия, требуемые для выборки команды из оперативной памяти и ее выполнения, называют **циклом команды**. В общем случае цикл команды включает следующие этапы:

- выборка команды из ОП;
- формирование адреса следующей команды;
- декодирование команды;
- вычисление адресов операндов;
- выборка операндов из ОП;
- исполнение операции;
- запись результата в память.

Упрощенно цикл команды последовательной ЭВМ можно представить в виде следующего алгоритма. Допустим выполняемая программа, состоящая из последовательности N команд K_1, K_2, …, K_i, …, K_N, загружена в оперативную память, и центральный процессор должен выполнить очередную команду K_i. Адрес этой команды A_i в этот момент времени находится в регистре программного счетчика.

1. В начале устройство управления вырабатывает сигнал на выборку в ЦП содержимого ячейки памяти по адресу A_i. Поскольку по этому адресу размещается очередная команда K_i, именно она и попадает в ЦП. Как правило, ЦП содержит

специальный **регистр команды** (РК), в котором и размещается полный код команды.

2. Значение программного счетчика увеличивается на длину команды K_i, и теперь он указывает на адрес A_{i+1} следующей, $(i + 1)$-й команды K_{i+1}.

3. После выборки команды ЦП определяет, какую операцию он должен выполнить по этой команде. Для этого УУ производит *декодирование* находящейся в РК команды. В результате декодирования выясняются следующие моменты:

- находится ли в РК полный код команды или требуется дозагрузка остальных слов команды;
- какие последующие действия нужны для выполнения данной команды;
- если команда использует операнды, то откуда они должны быть взяты (из регистров ЦП или из ОП);
- если команда формирует результат, то куда этот результат должен быть помещен.

Ответы на первые два вопроса дает *расшифровка кода операции*. Для случая аппаратного управления результатом расшифровки кода операции может являться *унитарный код*, каждый разряд которого соответствует одной из команд. При микропрограммном управлении результатом расшифровки может быть *адрес в памяти микропрограмм*, по которому находится первая микрокоманда микропрограммы для реализации указанной в команде операции.

4. Если на предыдущем этапе выясняется, что команда использует операнды, то осуществляется *вычисление их адресов*. Адрес каждого операнда вычисляется в соответствии с указанным в команде *способом адресации*.

5. Без ограничения общности допустим, что команда K_i производит некоторую операцию (например, сложение) над двумя операндами O_{i1} и O_{i2}. Расшифровав код операции сложения и вычислив адреса операндов O_{i1} и O_{i2}, устройство управления выдает сигналы на выборку из памяти в АЛУ сначала операнда O_{i1}, затем — операнда O_{i2}.

6. После того как оба операнда считаны из памяти, АЛУ производит требуемую операцию (сложение) и получает результат (сумму) R_i.

7. Устройство управления вырабатывает сигналы на запись результата R_i команды K_i в память (как вариант — в один из регистров ЦП). После этого центральный процессор переходит к выполнению команды K_{i+1}, т. е. к пункту 1 алгоритма.

Приведенный алгоритм обычно называют **стандартным циклом команды**. В классической последовательной ЭВМ отклонение от стандартного цикла команды происходит в двух случаях — при *изменении порядка выполнения команд программы* и при *операциях ввода-вывода*. Если в качестве *i*-й команды выступает команда перехода, вызова подпрограммы или возврата из подпрограммы, то пункты 1–3 стандартного цикла также выполняются. Но результатом выполнения команды в этом случае будет не результат операции, а соответствующее *изменение*

программного счетчика. Его новое значение будет равно адресу той команды, на выполнение которой следует перейти.

Для ввода-вывода информации в составе программы используются *специальные команды ввода-вывода*. Если на 3-м шаге цикла команды после декодирования выясняется, что очередная команда предписывает выполнить операцию ввода-вывода, то устройство управления активизирует соответствующий канал (порт) ввода-вывода, а через него — необходимое устройство ввода-вывода. Результатом такой команды будет либо считывание информации в центральную часть ЭВМ при вводе, либо запись информации на соответствующее устройство вывода.

2.3. СИСТЕМА КОМАНД ЭВМ

Как уже указывалось ранее, программа для ЭВМ состоит из определенного набора команд. Каждая команда состоит из определенного набора полей. На рис. 2.2 представлен обобщенный формат машинной команды.

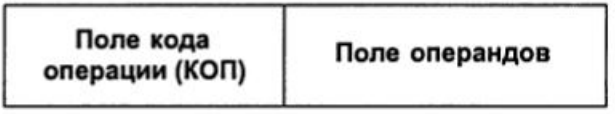


Рис. 2.2. Формат машинной команды

Каждая ЭВМ имеет свой набор определенных команд, который образует систему команд определенного типа (семейства) вычислительных машин.

Под **системой команд** будем понимать совокупность различных типов команд вместе с правилами их кодирования.

Команды ЭВМ можно классифицировать по различным признакам, в качестве которых можно использовать:

1) состав и сложность команд;
2) место хранения операндов;
3) способ обработки данных и тип операндов.

2.3.1. Классификация по составу и сложности команд

Если взять в качестве признака классификации **состав и сложность команд**, то можно выделить следующие системы команд:

- полный набор команд (архитектура CISC — Complex Instruction Set Computer);
- сокращенный набор команд (архитектура RISC — Reduced Instruction Set Computer);
- набор команд сверхбольшой длины (VLIW — Very Long Instruction Word).

ЭВМ с архитектурой CISC имеют в своем составе сложные команды, приближающиеся к операторам языков высокого уровня. Основоположником CISC-архитектур является компания IBM со своим семейством S/360. Этот подход

фирма продолжила и в последующих семействах S/370, S/390 и т. д. Эта же архитектура была реализована в советских ЭВМ семейств ЕС ЭВМ. Такой же подход использует и компания Intel для своих микропроцессоров архитектуры x86.

Для CISC-процессоров характерно:

- сравнительно небольшое число регистров общего назначения;
- большое количество машинных команд, некоторые из которых нагружены семантически аналогично операторам высокоуровневых языков программирования и выполняются за много тактов;
- большое количество методов адресации;
- большое количество форматов команд различной разрядности;
- преобладание двухадресного формата команд;
- наличие команд обработки типа «регистр-память», «память-память».

ЭВМ с архитектурой CISC имеют достаточно сложный аппаратный состав, главным образом УУ, что негативно сказывается на производительности.

Проведенные рядом авторов исследования показали, что доля команд, эквивалентных операторам языков высокого уровня, в объеме программ низка и не превышает 10–20%. В то же время объем аппаратных средств для их реализации весьма велик. Так, например, емкость памяти микропрограмм при поддержании сложных команд может увеличиваться на 60%.

Недостатки CISC-архитектур и привели к необходимости разработки ЭВМ с *сокращенным набором команд* или RISC-архитектур . Этот термин впервые был использован Д. Паттерсоном и Д. Дитцелем в 1980 г. Зачатки такой архитектуры уходят своими корнями к ЭВМ CDC6600, разработчики которой (Торнтон, Крэй и др.) осознали важность упрощения набора команд для построения быстрых вычислительных машин. Эту традицию упрощения архитектуры С. Крэй с успехом применил при создании широко известной серии суперкомпьютеров компании Cray Research. Однако окончательно понятие RISC в современном его понимании сформировалось на базе трех исследовательских проектов компьютеров: процессора 801 компании IBM, процессора RISC университета Беркли и процессора MIPS Стенфордского университета.

Разработка экспериментального проекта компании IBM началась еще в конце 70-х годов, но его результаты никогда не публиковались, и компьютер на его основе в промышленных масштабах не изготавливался. В 1980 г. Д. Паттерсон с коллегами из Беркли начали свой проект и изготовили две машины, которые получили названия RISC-I и RISC-II. Главными идеями этих машин было отделение медленной памяти от высокоскоростных регистров и использование регистровых окон. В 1981 г. Дж. Хеннесси со своими коллегами опубликовали описание стенфордской машины MIPS, основным аспектом разработки которой была *эффективная реализация конвейерной обработки* посредством тщательного планирования компилятором его загрузки. Эти три машины имели много общего. Все они придерживались архитектуры, отделяющей команды обработки от команд работы

с памятью, и делали упор на эффективную конвейерную обработку. Система команд разрабатывалась таким образом, чтобы выполнение любой команды занимало небольшое количество машинных тактов (предпочтительно один машинный такт). Сама логика выполнения команд с целью повышения производительности ориентировалась на аппаратную, а не на микропрограммную реализацию. Чтобы упростить логику декодирования команд, использовались команды фиксированной длины и фиксированного формата.

Среди других особенностей RISC-архитектур следует отметить наличие достаточно большого регистрового файла (в типовых RISC-процессорах реализуются 32 или большее число регистров по сравнению с 8–32 регистрами в CISC-архитектурах), что позволяет большему объему данных храниться в регистрах на процессорном кристалле большее время и упрощает работу компилятора по распределению регистров под переменные. Для обработки, как правило, *используются трехадресные команды*, что, помимо упрощения дешифрации, дает возможность сохранять большее число переменных в регистрах без их последующей перезагрузки.

Таким образом, для RISC-компьютеров характерными чертами являются:

- отделение команд обработки от команд работы с памятью;
- реализация конвейерной обработки данных;
- реализация как аппаратного, так и микропрограммного принципа управления;
- наличие большого числа регистров;
- использование преимущественно 3-х адресных команд;
- простота архитектуры.

Ко времени завершения университетских проектов (1983–1984) обозначился также прорыв в технологии изготовления сверхбольших интегральных схем. Простота архитектуры и ее эффективность, подтвержденная этими проектами, вызвали большой интерес в компьютерной индустрии, и с 1986 г. началась активная промышленная реализация архитектуры RISC. К настоящему времени эта архитектура прочно занимает лидирующие позиции на мировом компьютерном рынке рабочих станций и серверов.

Развитие архитектуры RISC в значительной степени определялось и прогрессом в области создания оптимизирующих компиляторов. Именно современная техника компиляции позволяет эффективно использовать преимущества большего регистрового файла, конвейерной организации и большей скорости выполнения команд. Современные компиляторы используют также преимущества другой оптимизационной техники для повышения производительности, обычно применяемой в процессорах RISC: реализацию задержанных переходов и суперскалярной обработки, позволяющей в один и тот же момент времени выдавать на выполнение несколько команд.

В предлагаемой архитектуре используется ограниченный набор команд, наиболее часто используемых в программах и оперирующих данными, размещенными только в регистрах процессора. Для обращения к памяти используются специальные команды чтения и записи.

Уменьшение количества форматов команд, способов адресации, отделение операций обработки данных от операций обращения к памяти позволило существенно упростить аппаратные средства ЭВМ и повысить их быстродействие.

В простейшем случае время выполнения любой программы можно оценить по следующему выражению:

$$T_{\text{вып}} = N_{\text{ком}} \cdot \tau_{\text{пр}} \cdot \eta_{\text{cpu}}, \qquad (2.1)$$

где $N_{\text{ком}}$ — число команд в программе; $\tau_{\text{пр}}$ — длительность такта процессора; η_{cpu} — среднее количество тактов процессора, приходящееся на одну команду.

В RISC-архитектурах сокращаются величины $\tau_{\text{пр}}$ и η_{cpu}.

Примерами RISC-архитектуры являются процессоры SPARC фирмы Sun и PowerPC фирмы IBM. Следует отметить, что в последнее время прослеживается тенденция к сближению архитектур CISC и RISC. Например, компания Intel в своих микропроцессорах использует автоматическую трансляцию CISC-кода в последовательность RISC-команд, которые выполняются RISC-ядром процессора.

Набор команд VLIW базируется на RISC-архитектуре. VLIW-машины выдают на выполнение фиксированное количество команд, которые сформатированы либо как одна большая команда, либо как пакет команд фиксированного формата. Планирование работы VLIW-машины всегда осуществляется компилятором.

Архитектура VLIW базируется на множестве независимых функциональных устройств. Вместо того чтобы пытаться параллельно выдавать в эти устройства независимые команды, в таких машинах несколько операций упаковываются в одну очень длинную команду. VLIW-команда может включать, например, две целочисленные операции, две операции с плавающей точкой, две операции обращения к памяти и операцию перехода. Такая команда будет иметь набор полей для каждого функционального устройства, возможно от 16 до 24 бит на устройство, что приводит к команде длиною от 112 до 168 бит.

Для машин с VLIW-архитектурой был разработан новый метод планирования выдачи команд, названный трассировочным планированием. При использовании этого метода из последовательности исходной программы генерируются длинные команды путем просмотра программы за пределами базовых блоков — линейных участков программы без ветвлений.

Компилятор с трассировочным планированием определяет участок программы без обратных дуг (переходов назад), который становится кандидатом для составления расписания. Обратные дуги обычно имеются в программах с циклами. Для увеличения размера тела цикла широко используется методика раскрутки циклов, что приводит к образованию больших фрагментов программы, не содержащих обратных дуг. Если имеется программа, содержащая только переходы вперед,

компилятор делает эвристическое предсказание выбора условных ветвей. Путь, имеющий наибольшую вероятность выполнения (его называют трассой), используется для оптимизации, проводимой с учетом зависимостей по данным между командами и ограничений аппаратных ресурсов. Во время планирования генерируется длинное командное слово. Все операции длинного командного слова выдаются одновременно и выполняются параллельно.

После обработки первой трассы планируется следующий путь, имеющий наибольшую вероятность выполнения (предыдущая трасса больше не рассматривается). Процесс упаковки команд последовательной программы в длинные командные слова продолжается до тех пор, пока не будет оптимизирована вся программа.

Ключевым условием достижения эффективной работы VLIW-машины является корректное предсказание выбора условных ветвей. Отмечено, например, что прогноз условных ветвей для научных программ часто оказывается точным.

Примерами VLIW-архитектуры являются процессоры Itanium фирмы Intel и процессоры семейства TMS320C62xx фирмы Texas Instruments.

2.3.2. Классификация по месту хранения операндов

По месту хранения операндов различают следующие наборы команд:

- стековый;
- аккумуляторный;
- регистровый;
- с выделенным доступом к памяти.

Выбор того или иного набора команд существенно влияет на следующие принципиальные моменты:

- сколько адресов будет в адресной части команды;
- какова будет длина этих адресов;
- как будет осуществляться доступ к операндам, и в конечном счете какой будет общая длина команды.

Рассмотрим эти четыре набора команд подробнее.

1. При использовании стека, последний организуется по принципу LIFO (Last In First Out — «последним вошел, первым вышел»). Для работы со стеком в основном используют две команды: **PUSH** (проталкивание данных в стек) и **POP** (выталкивание данных из стека). ЭВМ, система команд которых организована на основе стека, называют *стековыми*.

Операнды перед выполнением команды помещаются в две верхние ячейки стековой памяти. Результат операции заносится в стек. Для записи математических выражений используется форма записи, известная как *обратная польская запись* (нотация), предложенная польским математиком Я. Лукашевичем. Особенностью этой нотации является то, что в выражении отсутствуют скобки, а знак

операции располагается не между операндами, а следует за ними (постфиксная форма).

Например, выражение $X = Y + Z + X \cdot Y$ в обратной польской записи будет иметь вид:

$$X = YZ + XY * + \tag{2.2}$$

Такая форма записи позволяет однозначно определить порядок записи операндов и операций. К достоинствам системы команд на основе стека следует отнести возможность сокращения адресной части команд, так как все операции производятся через вершину стека, т. е. адреса операндов и результата в командах арифметической и логической обработки указывать не нужно. Поэтому код программы получается более компактным. Кроме того, при использовании стека достаточно легко реализуется декодирование команд.

В качестве недостатка следует отметить тот факт, что при использовании стека нельзя осуществлять произвольный доступ к памяти. Кроме того, стек становится «узким местом» в плане повышения производительности ЭВМ. Примерами ЭВМ, где реализована система команд на основе стека, могут служить машины фирмы Burroughs (B 5500, B 6500) и HP (HP 2116 B, HP 3000/70), а также транспьютеры (T212, T414, T800) фирмы Inmos.

Долгое время способ организации системы команд на основе стека считался неперспективным. Однако в связи с появлением языков **Java** и **Fort** семантика которых наиболее близка к стековой организации, интерес к стековой организации вновь возрастает.

2. Система команд **на базе аккумуляторов** исторически возникла одной из первых. В этом случае для хранения одного из операндов арифметической или логической операции в процессоре выделяется специально выделенный регистр, получивший название *аккумулятор*. В этот регистр чаще всего заносится и результат выполнения операции. Так как местоположение аккумулятора известно заранее, то в команде следует указывать адрес только одного операнда, что сокращает длину команды.

Изначально оба операнда находятся в ОП, и до выполнения операции один из них необходимо загрузить в аккумулятор, например с помощью команды **LOAD**. Результат операции может сохраняться в аккумуляторе, если он не является операндом для следующей команды. Для записи данных из аккумулятора в память используется команда записи, например **STORE**.

Достоинством системы команд с использованием аккумуляторов является уменьшение длины команды и простота декодирования. Эта система команд была использована в таких ЭВМ, как IBM 7090, PDP-8, MOS-6502.

3. Система команд **с использованием регистров** включает массив регистров, часто называемых *регистрами общего назначения* (*РОН*). По современным понятиям такой массив РОНов можно рассматривать как управляемую кэш-

память. Как правило, размер регистра совпадает с длиной машинного слова (8, 16, 32, 64 двоичных разрядов).

Количество РОНов зависит от архитектуры ЭВМ. Как уже отмечалось, в вычислительных машинах CISC-архитектуры их количество колеблется в диапазоне от 8 до 32, поэтому для указания номера регистра в команде используется до 5 разрядов. Исходя из этого, в команде можно одновременно указывать 2 или даже 3 номера регистра (номера регистров для хранения операндов и номер регистра результата).

В ЭВМ RISC-архитектуры количество РОНов существенно больше и может быть более сотни. Обычная длина таких регистров составляет 32 разряда. В командах могут использоваться до 3-х регистров одновременно.

Команды с использованием РОНов могут использовать операнды, находящиеся как в регистрах, так и в ОП. Поэтому в системе команд можно выделить подгруппы команд:

- регистр-регистр;
- регистр-память;
- память-память.

Команды типа «регистр-регистр» являются основными в RISC-машинах. В них также могут использоваться команды типа «регистр-память».

В CISC-машинах наиболее часто используются команды «регистр-память» и «память-память», каждый из этих подвидов системы команд имеет свои достоинства и недостатки.

При реализации системы команд с использованием РОНов можно выделить следующие достоинства:

- компактность объектного кода;
- высокая скорость вычислений (если не используется ОП).

Как недостаток следует отметить увеличенную, по сравнению с использованием аккумуляторов, длину команды.

В качестве ЭВМ с использованием системы команд на базе РОНов можно привести ЭВМ: IBM S/360/370/390, PDP-11, CDC 6600, все современные ПЭВМ.

4. Система команд **с выделенным доступом к памяти** предполагает для доступа к памяти использовать две специальные команды **LOAD** и **STORE**. Первая команда обеспечивает считывание (загрузку) данных из ОП в регистры ЦП, а вторая — занесение (сохранение) данных из регистров в ОП. Для обеспечения работы системы команд необходимы три шины. Две из них, расположенные между массивом РОНов и АЛУ, обеспечивают передачу в АЛУ операндов, хранящихся в РОНах, а третья служит для занесения результата в выделенный для этих целей регистр. Подобная система команд характерна почти для всех ЭВМ, имеющих RISC-архитектуру. В качестве примера можно привести следующие ЭВМ: PA-RISC (HP) , RS/6000 (IBM), SPARC (Sun), R4000 (MIPS), Alpha (DEC) и др.

Рис. 2.3. Классификация систем команд ЭВМ

На рис. 2.3 приведена классификация систем команд, использующая вышеуказанные признаки.

2.3.3. Классификация по способу обработки данных и типу операндов

Эта классификация, на наш взгляд, более удобна с точки зрения практики обучения. Особенно это проявляется при изучении языка ассемблера. Независимо от архитектуры ЭВМ все команды ЦП можно разделить на две большие группы:

1) команды привилегированные;
2) команды непривилегированные.

Первая группа, иногда называемая *командами, выполняющимися в режиме «супервизор/система»*, предоставляет пользователю возможность использовать все ресурсы ЭВМ.

Вторая группа, называемая *командами, выполняющимися в режиме «задача/пользователь»*, ограничивает использование некоторых возможностей ЭВМ.

Внутри каждой из этих групп можно выделить подгруппы, характеризующие вид обрабатываемой информации и выполняемые действия.

Так как в классической ЭВМ команды программы выполняются в порядке их расположения в памяти, то для получения адреса очередной команды достаточно увеличить содержимое программного счетчика на длину текущей команды. Для изменения такого порядка исполнения команд в ЭВМ включаются команды, позволяющие передать управление в другую точку программы. В адресной части таких команд содержится адрес точки перехода.

В современных ЭВМ можно выделить три типа команд, способных изменить последовательность вычислений:

- безусловные переходы;
- условные переходы (ветвления);
- вызовы подпрограмм (процедур) и возвраты из них.

В английском языке для указания команд *безусловного перехода*, как правило, используется термин **jump**, а для команд *условного перехода* — термин **branch**, хотя разные фирмы необязательно придерживаются этой терминологии. Например, компания Intel использует термин **jump** и для условных, и для безусловных переходов.

Частота использования этих команд по статистике примерно следующая. В программах доминируют команды условного перехода. Среди указанных команд управления на разных программах частота их использования колеблется от 66 до 78%. Следующие по частоте использования — команды безусловного перехода (от 12 до 18%). Частота переходов на выполнение процедур и возврата из них составляет от 10 до 16%.

При этом примерно 90% команд безусловного перехода выполняются относительно программного счетчика. Для команд перехода адрес перехода должен быть всегда заранее известен. Это не относится к адресам возврата, которые не известны во время компиляции программы и должны определяться во время ее работы.

Переход можно организовать следующими способами:

1) переход относительно текущего значения программного счетчика;
2) переход и возврат по косвенному адресу.

Наиболее простой способ определения адреса перехода заключается в указании его положения относительно текущего значения программного счетчика (с помощью смещения в команде), и такие переходы *называются переходами относительно программного счетчика*. Преимуществом такого метода адресации является то, что адреса переходов, как правило, расположены недалеко от текущего адреса выполняемой команды и указание относительно текущего значения программного счетчика требует небольшого количества бит в смещении. Кроме того, использование адресации относительно программного счетчика позволяет программе выполняться в любом месте памяти, независимо от того, куда она была загружена, т. е. этот метод адресации позволяет автоматически создавать перемещаемые программы.

Реализация возвратов и переходов по **косвенному адресу**, в которых адрес не известен во время компиляции программы, требует методов адресации, отличных от адресации относительно программного счетчика. В этом случае адрес перехода должен определяться динамически во время работы программы. Наиболее простой способ заключается *в указании регистра для хранения адреса возврата*.

Один из ключевых вопросов реализации команд перехода состоит в том, насколько далеко целевой адрес перехода находится от самой команды перехода

(глубина перехода)? И на этот вопрос статистика использования команд дает ответ: в подавляющем большинстве случаев переход идет в пределах 3–7 команд относительно команды перехода, причем в 75% случаев выполняются переходы в направлении увеличения адреса, т. е. вперед по программе.

Поскольку большинство команд управления составляют команды условного перехода, важным вопросом реализации архитектуры является определение **условий перехода**. Для этого используются три различных подхода.

При первом из них в архитектуре процессора предусматривается специальный регистр, разряды которого соответствуют определенным кодам условий. Команды условного перехода проверяют эти условия в процессе своего выполнения. Преимуществом такого подхода является то, что иногда установка кода условия и переход по нему могут быть выполнены без дополнительных потерь времени, что, впрочем, бывает достаточно редко. А недостатки такого подхода в том, что, во-первых, появляются новые состояния машины, за которыми необходимо следить (скрывать при прерывании и восстанавливать при возврате из него). Кроме того, и что очень важно для современных высокоскоростных конвейерных архитектур, коды условий ограничивают порядок выполнения команд в потоке, поскольку их основное назначение заключается в передаче кода условия команде условного перехода.

Второй метод заключается в простом использовании произвольного регистра (возможно одного выделенного) общего назначения. В этом случае выполняется проверка состояния этого регистра, в который предварительно помещается результат операции сравнения. Недостатком этого подхода является необходимость выделения в программе для анализа кодов условий специального регистра.

Третий метод предполагает объединение оператора сравнения и перехода в одной команде. Недостатком такого подхода является то, что эта объединенная команда довольно сложна для реализации (в одной команде надо указать и тип условия, и константу для сравнения, и адрес перехода). Поэтому в таких машинах часто используется компромиссный вариант, когда для некоторых кодов условий, например, для сравнения с нулем, используются объединенные команды, а для более сложных — регистр условий. Часто для анализа результатов команд сравнения для целочисленных операций и для операций с плавающей точкой используется разная техника, хотя это можно объяснить и тем, что в программах количество переходов по условиям выполнения операций с плавающей точкой значительно меньше общего количества переходов, определяемых результатами работы целочисленной арифметики.

Одним из наиболее заметных свойств большинства программ является преобладание в них сравнений на условие «равно/неравно» и сравнений с нулем. Поэтому в ряде архитектур такие команды выделяются в отдельный поднабор, особенно при использовании команд типа «сравнить и перейти». Переход выполняется, если истинным является условие, которое проверяет команда условного перехо-

да. В этом случае выполняется переход на адрес, заданный командой перехода. Поэтому все команды безусловного перехода всегда выполняемые. По статистике оказывается, что переходы назад по программе в большинстве случаев используются для организации циклов, причем примерно 60% из них составляют выполняемые переходы. В общем случае поведение команд условного перехода зависит от конкретной прикладной программы, однако иногда сказывается и зависимость от компилятора. Такие зависимости от компилятора возникают вследствие изменений потока управления, выполняемого оптимизирующими компиляторами для ускорения выполнения циклов.

Вызовы процедур и возвраты предполагают передачу управления и, возможно, сохранение некоторого состояния. Как минимум, необходимо сохранять адрес возврата. Некоторые архитектуры предлагают аппаратные механизмы для сохранения состояния регистров, в других случаях предполагается вставка в программу команд самим компилятором. Имеются два основных вида соглашений относительно сохранения состояния регистров. *Сохранение вызывающей (caller saving) программой* означает, что вызывающая процедура должна сохранять свои регистры, которые она хочет использовать после возврата в нее. *Сохранение вызванной процедурой* предполагает, что вызванная процедура должна сохранить регистры, которые она собирается использовать. Имеются случаи, когда должно использоваться сохранение вызывающей процедурой для обеспечения доступа к глобальным переменным, которые должны быть доступны для обеих процедур.

2.4. ФОРМАТЫ КОМАНД

Уже отмечалось (см. рис. 2.2), формат команды определяет ее структуру, т. е. количество двоичных разрядов, отводимых под команду, а также количество и расположение отдельных полей команды.

При разработке системы команд обычно учитывают следующие факторы:

- общее число команд, входящих в систему;
- общую длину команды;
- тип полей команды и их длину;
- простоту декодирования;
- адресуемость и способы адресации;
- стоимость оборудования для декодирования и исполнения команд.

Следует отметить, что длина команды существенно влияет на организацию и емкость ОП, структуру шин, сложность и быстродействие ЦП. Чем больше команд входит в систему команд и чем они длиннее, тем можно больше использовать способов адресации, большее число кодов операций, можно обращаться к большему объему ОП. Однако такой подход требует выделения большего количества разрядов под каждое поле команды, что, в свою очередь, приводит к увеличению времени выборки команды, расширению шины и, как результат, — усложнению аппаратуры.

На практике длину команды выбирают кратной байту, так как в большинстве современных ЭВМ ОП построена по байтному принципу. В связи с использованием различных способов адресации в систему команд включают команды различной длины.

Общая длина команды L_k может быть описана следующим соотношением:

$$L_k = \sum_{i=1}^{n} R_i + R_{\text{КОП}} + R_{\text{СА}}, \tag{2.3}$$

где n — количество адресов в команде; R_i — количество разрядов для записей i-го операнда; $R_{\text{КОП}}$ — разрядность поля для кода операции (КОП); $R_{\text{СА}}$ — разрядность поля способа адресации (СА).

Количество разрядов под поле для КОП выбирают так, чтобы можно было представить любую из операций. Если система команд включает $N_{\text{КОП}}$ различных операций, то минимальная разрядность поля под КОП $R_{\text{КОП}}$ определяется выражением:

$$R_{\text{КОП}} = \text{int}(\log_2 N_{\text{КОП}}), \tag{2.4}$$

где int — округление в большую сторону до целого числа.

С целью нахождения оптимального соотношения между полями под КОП и адресную часть, иногда поле КОП делают переменным. Изначально под поле КОП выделяют фиксированное число разрядов, но для отдельных команд длина поля может увеличиваться за счет добавления определенного числа разрядов из адресного поля.

Адресная часть команды содержит информацию о местонахождении данных и месте хранения результата операции. Адреса операндов и результата задаются указанием либо номеров регистров, либо соответствующими именами, обозначающими номера ячеек ОП.

Разрядность полей R_i и $R_{\text{СА}}$ можно рассчитать по нижеследующим формулам:

$$R_i = \text{int}(\log_2 N_i), \tag{2.5}$$

$$R_{\text{СА}} = \text{int}(\log_2 N_{\text{СА}}), \tag{2.6}$$

где N_i — количество ячеек ОП, к которому можно обратиться с помощью i-го адреса; $N_{\text{СА}}$ — количество способов адресации.

Таким образом,

$$L_k = \sum_{i=1}^{n} \text{int}(\log_2 N_i) + \text{int}(\log_2 N_{\text{КОП}}) + \text{int}(\log_2 N_{\text{СА}}). \tag{2.7}$$

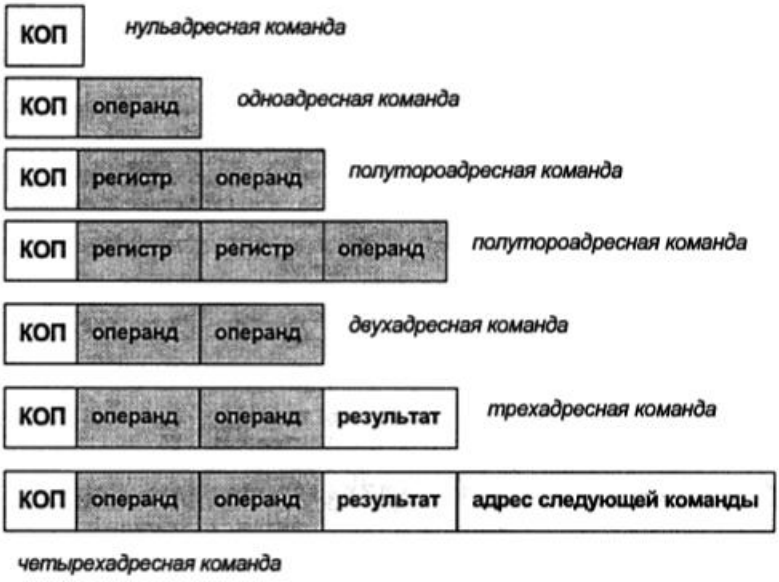


Рис. 2.4. Форматы команд

Количество адресов в команде в различных ЭВМ различно. В общем случае можно использовать следующие форматы команд:

- нульадресный;
- одноадресный;
- полуторадресный;
- двухадресный;
- трехадресный;
- четырехадресный.

На рис. 2.4 представлены все перечисленные выше форматы команд.

Команда «КОП/регистр/регистр/операнд» может быть одноадресной или полутороадресной в зависимости от длины поля, отводимого для размещения номера регистра.

Четырехадресные форматы команд применялись в первые годы создания ЭВМ. Одной из таких ЭВМ была EDVAC. Трехадресные форматы часто использовались в ЭВМ 60-х годов. В настоящее время наиболее употребительны двухадресные команды, хотя в системе команд ЭВМ могут быть и другие форматы, кроме четырехадресных.

При выборе того или иного формата команд учитывают такие факторы, как:

- емкость ОП, требуемой для хранения программы;
- время выполнения программы;
- эффективность использования ячеек ОП при хранении программы.

На рис. 2.5 приведен обобщенный формат команд ЭВМ с МП архитектуры x86 (Intel, AMD).

Рис. 2.5. Обобщенный формат команд ЭВМ архитектуры x86

Как следует из рис. 2.5, формат команды является переменным. В младших моделях длина команды составляет от 1 до 6 байтов, в старших длина команды может иметь длину до 11 байтов.

В подполе префиксов размещается следующая информация:

- префикс команды (повторения или блокировки доступа к шине данных);
- префикс переопределения сегмента;
- префикс переопределения размера операнда (для старших моделей);
- префикс переполнения размера адреса (для старших моделей).

При выборе того или иного формата команды учитывают время выполнения команды, которое складывается из времени непосредственно выполнения команды и времени обращения к памяти.

При использовании одноадресных команд необходимы два обращения к памяти: выборка команды и выборка операнда.

При выполнении же трехадресной команды таких обращений нужно четыре: выборка команды, выбора 1-го операнда, выборка 2-го операнда, запись в память результата.

В общем случае время выполнения программы можно оценить по следующему соотношению:

$$T = N_1 \cdot \tau_{1A} + N_2 \cdot \tau_{2A}, \tag{2.8}$$

где N_1 — количество арифметических и логических команд в программе; τ_{1A} — время выполнения одной арифметической или логической команды; N_2 — количество неарифметических команд; τ_{2A} — время выполнения одной неарифметической команды; $A = \{1, 2, 3\}$ — индекс, определяющий количество адресов в команде.

В свою очередь N_2 можно представить как

$$N_2 = N_{21} + N_{22},$$

где N_{21} — количество команд передачи управления (их число не зависит от адресности); N_{22} — количество вспомогательных команд пересылок данных в регистр сумматора и из него.

Времена τ_{1A} и τ_{2A} складываются из времени выборки команды из памяти τ_0 (время на одно обращение к памяти) и времени считывания/записи данных $A\tau_0$.

В случае арифметической команды следует также учитывать время выполнения арифметической операции τ_1.

Таким образом, имеем:

$$\tau_{1A} = \tau_1 + \tau_0 + A \cdot \tau_0 = \tau_1 + (A+1)\,\tau_0, \tag{2.9}$$

$$\tau_{2A} = \tau_0 + A \cdot \tau_0 = (A+1) \cdot \tau_0. \tag{2.10}$$

Тогда выражение (2.8) примет вид:

$$T = N_1 \cdot [\tau_1 + (A+1) \cdot \tau_0] + (N_{21} + N_{22}) \cdot (A+1) \cdot \tau_0. \tag{2.11}$$

Если подставить в (2.10) значения $A = 1$ для одноадресной и $A = 3$ для трехадресной команды, то можно получить величину ΔT — разность времени на реализацию программы с помощью одноадресных и трехадресных команд, принимая во внимание, что для трехадресных команд $N_{22} = 0$.

$$T_1 = N_1 \cdot (\tau_1 + 2\tau_0) + (N_{21} + N_{22}) \cdot 2\tau_0 =$$

$$= N_1 \cdot \tau_1 + 2N_1 \cdot \tau_0 + 2\tau_0 \cdot N_{21} + 2\tau_0 \cdot N_{22};$$

$$T_3 = N_1 \cdot (\tau_1 + 4\tau_0) + (N_{21} + N_{22}) \cdot 4\tau_0 =$$

$$= N_1 \cdot \tau_1 + 4N_1 \cdot \tau_0 + 4\tau_0 \cdot N_{21}.$$

$$\Delta T = T_1 - T_3 =$$

$$= N_1 \cdot \tau_1 + 2N_1 \cdot \tau_0 + 2\tau_0 \cdot N_{21} + 2\tau_0 \cdot N_{22} - N_1 \cdot \tau_1 - 4N_1 \cdot \tau_0 - 4\tau_0 \cdot N_{21} =$$

$$= -2N_1 \cdot \tau_0 - 2\tau_0 \cdot N_{21} + 2\tau_0 \cdot N_{22} = 2\tau_0 \cdot (N_{22} - N_1 - N_{21}).$$

Таким образом,

$$\Delta T = 2\tau_0\,(N_{22} - N_1 - N_{21}). \tag{2.12}$$

Двухадресные команды в плане времени выполнения программы занимают промежуточное положение между одноадресными и трехадресными.

При реализации последовательных алгоритмов одноадресные команды выгодно использовать в тех программах, где больше арифметических и логических команд.

При реализации параллельных алгоритмов, если количество параллельных операций больше количества операций передачи управления, выгоднее использовать трехадресные команды.

2.5. СПОСОБЫ АДРЕСАЦИИ

Вопрос о том, каким образом в адресном поле команды может быть указано местоположение операндов, является главным при создании любой ЭВМ. С точки

зрения сокращения аппаратных средств очевидно стремление к уменьшению длины адресного поля в команде. Однако в этом случае программирование на языках низкого уровня весьма отлично от программирования на языках высокого уровня (ЯВУ). С целью сближения конструкций операторов ЯВУ и языков низкого уровня целесообразно иметь большое адресное пространство.

Эти противоречия и привели к тому, что в системе команд ЭВМ предусматриваются несколько различных способов адресации.

При рассмотрении различных способов адресации будем использовать следующие понятия.

1. **Исполнительный адрес операнда ($A_{исп}$)** — двоичный код номера ячейки памяти, служащей источником или приемником операнда. Этот код подается на адресные входы ОП, и по нему происходит фактическое обращение к указанной ячейке.

2. **Адресный код команды ($A_к$)** — двоичный код в адресном поле команды, из которого необходимо сформировать исполнительный адрес операнда. Как правило, $A_{исп}$ и $A_к$ не совпадают, и для доступа к данным необходимо соответствующее преобразование, которое осуществляется с помощью способа адресации.

3. **Способ адресации** — *это метод формирования исполнительного адреса по адресному коду команды.*

Различные способы адресации по-разному влияют на процесс обработки информации. Одни способы адресации позволяют увеличить емкость адресуемой памяти без увеличения длины команды, но снижают скорость выполнения команды. Другие способы ускоряют операции над массивами данных, третьи — упрощают работу с подпрограммами и т. п.

В современных ЭВМ для выполнения одной и той же операции используют разные способы адресации. Часто различным способам адресации соответствуют различные коды операций. В некоторых случаях в команде может быть несколько таких полей. Возможна ситуация, когда в команде вообще нет адресной информации, — это так называемая **неявная адресация**.

Рассмотрим наиболее часто используемые способы адресации.

Во всех типах современных ЭВМ используется байтовая организация памяти. Байт является минимальной адресуемой частью памяти, состоящей из 8 разрядов (битов). Биты нумеруются справа налево от 0 до 7. Кроме байта, используются более крупные единицы измерения памяти: Кбайт (2^{10} = 1 024 байт), Мбайт (2^{20} = 1 048 576 байт), Гбайт (2^{30} = 1 073 741 824 байта), Тбайт (2^{40} = 1 099 511 627 776 байт).

Каждый байт памяти имеет свой номер, называемый **адресом**. Кроме байта, адресовать в памяти можно и более крупные единицы памяти. Рассмотрим эти единицы.

Слово — совокупность нескольких смежных байтов, общее число битов в которых равно разрядности ЭВМ. Например, для 32-разрядных машин слово будет составлять 4 байта, для 64-разрядных — 8 байт.

Поле — непрерывная по возрастающим значениям адресов последовательность байтов в оперативной памяти. За адрес поля принимается адрес крайнего левого байта поля. Использование поля той или иной длины определяется типом команды, адресующейся к этому полю.

Параграф (только для ЭВМ с архитектурой x86) — совокупность 16 смежных байтов ОП. Адрес параграфа должен быть кратен 16.

Страница — некоторая совокупность байтов. Величина ее зависит от класса ЭВМ. Так, для ЭВМ типа PDP-11 размер страницы являлся переменным и колебался от 64 байт до 8 Кбайт. Применительно к видеопамяти (экран монитора) под страницей понимается содержимое экрана.

Сегмент — применительно к ЭВМ с архитектурой x86 — это область ОП, начинающаяся с границы параграфа и имеющая размер до 64 Кбайт ($2^{16} = 65\,536$ бит). В старших моделях этих ЭВМ размер сегмента может достигать 4 Гбайт. Для ряда ЭВМ понятия «сегмент» и «страница» являются эквивалентными.

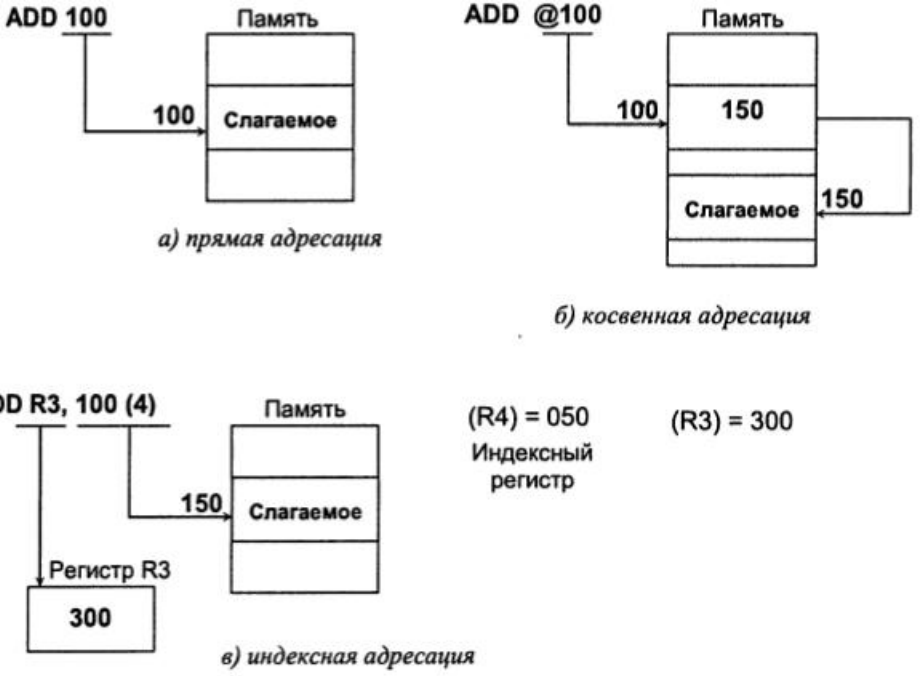


Рис. 2.6. Способы адресации

Наибольшее распространение получили следующие способы адресации: *непосредственная*, *прямая*, *косвенная* и *индексная*. Схематично три последних способа изображены на рис. 2.6 (*а*, *б*, *в*).

При **непосредственной адресации** сам операнд содержится в адресном поле команды. Этот способ, как правило, используется при выполнении арифметических операций, операций сравнения и для загрузки констант в регистры.

При **прямой адресации** в адресном поле команды указывается адрес операнда в памяти (либо номер регистра, содержащего операнд). На рис.2.6, *а* операнд (слагаемое) размещен по адресу **100**. Этот адрес и указан в команде сложения **ADD 100**.

При **косвенной адресации** в адресном поле команды указывается адрес ячейки памяти или номер регистра, где содержится адрес операнда. Для выборки операнда необходимо двукратное обращение к памяти — сначала извлекается адрес операнда, а затем — сам операнд. На рис.2.6, *б* адрес операнда (150) расположен по адресу **100**, который и указывается в команде **ADD @100**.

При **индексной адресации** к формируемому исполнительному адресу прибавляется значение специального регистра, называемого *индексным*. На рис. 2.6, *в* показан процесс формирования исполнительного адреса при индексной адресации. В качестве индексного регистра используется **R4**, содержащий число **50**. Это число добавляется к адресу **100**, указанному в поле команды. По получившемуся исполнительному адресу **150** считывается операнд, который складывается с содержимым регистра **R3 (300)**.

Рассмотрим, как часто используются наиболее распространенные способы адресации. Естественно, что частота использования тех или иных способов адресации существенно зависит от реализованной на ЭВМ системы команд.

Так, например, непосредственная адресация предполагает, что если операнд является числом, то оно представляется в дополнительном коде. Если для размещения операнда используется регистр, разрядность которого превышает длину непосредственного операнда, то операнд размещается в младшей части регистра.

Существенным недостатком непосредственной адресации является не только то, что в качестве операнда можно использовать только константы, но и то, что размер операнда ограничен длиной адресного поля команды, которое в большинстве случаев меньше длины машинного слова.

Проведенные рядом авторов исследования показывают, что в 50–60% команд с непосредственной адресацией длина операнда не превышает 8 бит, а в 75–80% — 16 бит. Средний процент использования непосредственной адресации по всем командам составляет 35% для целочисленных вычислений и 10% — при обработке чисел с ПТ.

С точки зрения эффективности непосредственная адресация является «идеальным» способом адресации, с точки зрения стоимости оборудования и временных затрат — не самым лучшим.

Прямой способ адресации имеет существенный недостаток, связанный с ограниченным размером адресного пространства. Кроме того, адрес, указный в команде, не может быть изменен в процессе вычислений, что накладывает ограничение по произвольному размещению программы в памяти.

При косвенной адресации содержимое адресного поля команды остается неизменным, в то время как косвенный адрес в процессе выполнения команды можно изменять. Использование косвенной адресации упрощает обработку массивов, списков и передачу параметров подпрограмм. Недостатком этого способа адресации является необходимость в двукратном обращении к памяти: сначала для извлечения адреса операнда, а затем для обращения к операнду. Кроме того, задействуется лишняя ячейка памяти для хранения исполнительного адреса операнда.

Разновидностью прямой и косвенной адресации является **регистровая**, когда адресное поле команды указывает не на ячейку памяти, а на регистр. При **прямой регистровой** адресации операнд непосредственно размещается в регистре (на рис. 2.6 *в*, приведен пример прямой регистровой адресации — первый операнд, число **300**, размещается в регистре **R3**), при **косвенной регистровой** — регистр содержит адрес операнда в памяти. Однако широкие возможности по использованию регистрового способа адресации ограничиваются малым числом РОНов в составе процессора.

Существует достаточное количество способов адресации с использованием **базы** и **смещения**. В этом случае исполнительный адрес формируется путем сложения двух компонент — **базового адреса** и **смещения**. Как правило, базовый адрес загружается в специальный **регистр базы**. Для определения смещения используются уже перечисленные способы адресации — непосредственная, прямая и косвенная, очень часто в сочетании с индексным способом. Главное достоинство адресации с базой и смещением заключается в том, что программа становится легко перемещаемой и может быть загружена практически в любую область памяти. Для перемещения программы из одной области памяти в другую достаточно изменить значение базового адреса.

Следует заметить, что для ЭВМ с использованием РОНов затруднительно сделать однозначный вывод о предпочтительности того или иного способа адресации. Можно отметить только, что интенсивность использования способа адресации напрямую зависит от характера решаемой задачи.

2.6. СИСТЕМА ПРЕРЫВАНИЙ ПРОГРАММ ЭВМ

2.6.1. Необходимость системы прерываний

Нетрудно заметить, что рассмотренный нами упрощенный алгоритм цикла команды последовательной ЭВМ является крайне неэффективным с точки зрения как производительности и полноты использования аппаратных ресурсов, так и организации вычислений. Рассмотрим основные причины низкой эффективности

и соответствующие способы оптимизации функционирования последовательных ЭВМ.

Первой причиной возможной потери эффективности является *различное быстродействие* разных компонент ЭВМ. Наиболее быстрым устройством в ЭВМ традиционно является центральный процессор. Уже оперативная память имеет в разы меньшее быстродействие, не говоря об устройствах ввода-вывода, быстродействие которых может отличаться от трех до семи порядков по сравнению с центральным процессором. Различие в быстродействии неизбежно приводит к тому, что более производительные компоненты ЭВМ начинают простаивать в ожидании окончания работы менее производительными.

Например, ЦП будет неизбежно простаивать во время выполнения обычной арифметической операции в ожидании окончания взаимодействия с ОП. Напомним, что в цикле команды этапы выборки команды, выборки операндов, записи результата требуют информационных обменов с ОП. Учитывая кратное расхождение в быстродействии между ЦП и ОП, получаем, что процессор будет простаивать большую часть времени.

Заметим, что еще менее эффективным будет выполнение команды ввода-вывода. Например, при считывании слова из устройства ввода ЦП будет вынужден ожидать окончания операции передачи данных из УВВ. Соотношение по быстродействию между ЦП и УВВ может быть таково, что для процессора ожидание окажется фактически «машинной бесконечностью».

Таким образом, *непосредственная реализация алгоритма стандартного цикла команды последовательной ЭВМ приведет к тому, что производительность ЭВМ будет определяться быстродействием самого медленного из ее элементов.*

Другой причиной низкой эффективности является то, что алгоритм стандартного цикла является *замкнутым* и не предполагает какого-либо *внешнего по отношению к нему воздействия*. Вычислительный процесс в этом случае представляет собой последовательность выполнения команд программы, причем эта последовательность определяется исключительно самой программой и изначальными входными данными.

Нетрудно видеть, что в приведенной последовательной модели отсутствует понятие события и соответственно *обработки событий*. В то же время работу ЭВМ можно представить как последовательность *программно-определяемых* и *программно-независимых* событий. Программно-определяемые события порождаются самой программой (например, обнуление счетчика цикла), возможные моменты их появления известны и учтены в программе. Программно-независимые события вызываются посторонними от программы источниками (например, нажатие клавиши оператором), моменты их возникновения заранее неизвестны и не могут быть предопределены программой.

События, происходящие вне процессора, как правило, программно-независимые (выход параметров объекта за дозволенные пределы, прием сетевого пакета и т. п.) и происходят асинхронно. События, происходящие внутри процессора, могут быть двух типов. Последовательность арифметических операций определяется программой, в то время как особые ситуации (переполнение, попытка деления на нуль и т. д.) зависят от сочетания операндов и предусмотреть их во время программирования практически невозможно.

ЭВМ, вообще говоря, должна реагировать на любые события, которые могут повлиять на процесс вычислений. В случае программно-определяемых событий для этого достаточно иметь специальный набор команд (переход по нулю, по знаку и т. д.), обработка подобных событий не выходит за рамки стандартного цикла команды. Если же события программно-независимые, то алгоритм стандартного цикла, как правило, становится крайне неэффективным. Единственным возможным решением по обработке программно-независимых событий в рамках стандартного алгоритма является периодическая проверка наступления того или иного события с использованием специальной команды перехода.

Пусть в системе команд имеется команда перехода по какому-нибудь программно-независимому событию. Исходя из определения, во-первых, неизвестно, когда это событие произойдет, во-вторых, неясно, как часто следует вставлять в программу команду перехода по данному событию. Например, если вставка будет производиться каждые 25 команд, то объем памяти для программы увеличивается на 4%, а реакция на событие может произойти с большой задержкой (максимум через 25 команд). Если в программе потребуется обработка нескольких программно-независимых событий, это приведет к значительному увеличению объема программы, времени ее выполнения и реакции на события и, кроме этого, сильно затруднит работу программиста.

Для нивелирования разницы быстродействия между компонентами последовательной ЭВМ в алгоритм ее функционирования вносятся элементы *параллельной обработки информации*, заключающейся в **совмещении во времени** различных операций стандартного цикла команды.

Совмещение во времени работы ЦП и информационных обменов с ОП, как правило, организуется по принципу конвейера. Например, пока в АЛУ процессора выполняется очередная команда, устройство управления может производить выборку из ОП следующей команды. Более подробно конвейерная обработка информации рассматривается в п. 5.5. Кроме этого, для оптимизации работы ОП применяются различные специальные подходы — иерархическая организация, оптимизация доступа к оперативным запоминающим устройствам (ОЗУ), структурные методы повышения быстродействия ОП (подробнее см. гл. 3).

Совмещение во времени работы ЦП и УВВ достигается за счет применения различных *методов управления вводом-выводом* (п. 4.1.2), смысл которых сводится к тому, что центральный процессор не дожидается окончания операций ввода-

вывода, а лишь инициирует их и обрабатывает их завершение. Значительный промежуток времени между началом и окончанием операции ввода-вывода может быть потрачен процессором на выполнение вычислительной работы.

Нетрудно заметить, что хотя процессор и начинает каждую операцию ввода-вывода, момент ее завершения, зависящий от множества факторов (объем передаваемых данных, быстродействие УВВ), заранее неизвестен и может быть отнесен к программно-независимым событиям. Таким образом, решение проблемы различного быстродействия компонент ЭВМ также зависит от эффективности обработки программно-независимых событий.

Чтобы ЭВМ могла реагировать на программно-независимые события при минимальных усилиях программиста и максимально возможном быстродействии, ее структура дополняется специальными аппаратно-логическими средствами, совокупность которых называют **системой прерывания программ** (СПП).

2.6.2. Общая организация системы прерываний

Появление системы прерывания в компьютерах конца 50-х годов позволило существенно продвинуться в разработках ЭВМ, по-новому переосмыслив их возможности и среды применения. Действительно, уже на первых ЭВМ, обладающих этим свойством, появилась возможность автономной работы устройств ввода-вывода после их запуска центральным процессором. По окончании работы или в связи с другой причиной устройство ввода-вывода могло через прерывание заставить центральный процессор обратить на себя внимание. Это открыло путь к широкому совмещению операций, что повлекло за собой естественное усложнение математического обеспечения. Современные операционные системы полностью разрабатываются и базируются на развитой системе прерываний.

Суть работы любой системы прерываний сводится к следующему. При возникновении события, требующего немедленной реакции со стороны ЭВМ, ЦП прекращает обработку текущей программы и переходит к выполнению другой программы, специально предназначенной для данного события, по завершении которой возвращается к выполнению отложенной программы. Такой режим работы называется **прерыванием**.

Каждое событие, требующее прерывания, сопровождается специальным сигналом, который называется **запросом прерывания**. Программа, затребованная запросом прерывания, называется **обработчиком прерывания**, или **прерывающей программой.** Каждому запросу прерывания в ЭВМ присваивается свой номер (**тип прерывания**), используемый для определения адреса обработчика прерывания.

Запросы на прерывание могут возникать из-за сбоев в аппаратуре, зафиксированных схемами контроля, переполнения разрядной сетки, деления на нуль, выхода за установленные для данной программы области памяти, затребования внешним устройством операции ввода-вывода, завершения этой операции ввода-

вывода или возникновения при этой операции особых условий и т. д. Некоторые из этих запросов порождаются самой программой, но время их возникновения невозможно предсказать заранее.

По окончании работы обработчика прерывания переход может быть осуществлен либо к прерванной программе, либо к другой прерывающей программе. Так как всевозможные запросы на прерывание вырабатываются независимо и асинхронно, то возможны такие ситуации:

- приход запросов последовательный;
- одновременный приход нескольких запросов;
- приход запроса во время выполнения обработчика прерывания.

Поступившие запросы должны обрабатываться в определенном порядке. Если в ЭВМ имеются средства для обслуживания запросов в порядке присвоенного им приоритета, то такие системы прерывания называются **приоритетными**.

Прерывания можно разделить на *аппаратные*, *логические* и *программные*.

Аппаратные прерывания вырабатываются аппаратурой ЭВМ или устройствами ввода-вывода с целью переключить внимание процессора на обработку того или иного внешнего события. Примерами аппаратных прерываний могут служить сигнал о сбое питания, сигналы от внешних устройств — клавиатуры, модема, принтера и др.

Запросы на **логические прерывания** вырабатываются внутри процессора при появлении исключительных ситуаций: при попытке деления на нуль, при переполнении разрядной сетки арифметико-логического устройства, при нарушении защиты памяти; при достижении программой одной из контрольных точек и т. п. Общим для аппаратных и логических прерываний является то, что запрос на них вырабатывается в виде специального электрического сигнала. Аппаратные и логические прерывания определенных типов могут быть запрещены в определенные моменты времени, т. е. запрос на прерывание запрещенного типа будет проигнорирован системой. Запрещенные в данный момент времени прерывания называются **замаскированными**.

Программные прерывания вырабатываются самой программой путем выполнения специальной *команды вызова прерывания*. Программные прерывания используются для обращения прикладной программы к сервисным функциям операционной системы (чтение-запись файла, взаимодействие с различными устройствами ввода-вывода, межпроцессное взаимодействие и т. п.).

При выполнении программы после каждого рабочего такта процессора изменяются содержимое регистров, счетчиков, состояние отдельных управляющих триггеров, т. е. изменяется состояние процессора. Информация о состоянии процессора лежит в основе многих процедур управления вычислительным процессом. Например, важнейшим элементом является значение программного счетчика, определяющее адрес следующей команды выполняемой программы. Информация, без которой невозможно продолжение работы процессора, должна

сохраняться при каждом переключении процессора с прерываемой программы на обработчик прерывания и обратно.

Совокупность значений наиболее существенных информационных элементов называется **вектором состояния, или словом состояния процессора** — ССП (в некоторых случаях ССП называют **словом состояния программы**).

ССП в каждый момент времени должно содержать информацию, достаточную для продолжения выполнения программы или повторного пуска ее с точки, соответствующей моменту формирования данного вектора. ССП формируется в соответствующем регистре процессора или в группе регистров, которые могут использоваться и для других целей. ССП, как минимум, включает в себя текущие значения программного счетчика и регистра флагов программы.

При поступлении запроса прерывания ЭВМ выполняет следующую последовательность действий:

1) определение наиболее приоритетного незамаскированного запроса на прерывание (если одновременно поступило несколько запросов);
2) определение типа выбранного запроса;
3) сохранение слова состояния текущей (прерываемой) программы;
4) определение адреса обработчика прерывания и передача управления первой команде этого обработчика;
5) выполнение программы-обработчика прерывания;
6) восстановление сохраненного ССП прерванной программы;
7) продолжение выполнения прерванной программы.

Этапы 1–4 выполняются аппаратными средствами ЭВМ автоматически при появлении запроса прерывания. Этап 6 также выполняется аппаратно по команде возврата из обработчика прерывания. Обработчик прерывания, как и любая другая программа, пишется человеком-программистом.

Рис. 2.7. Временная диаграмма прерывания

Рассмотрим временную диаграмму прерывания (рис. 2.7). Запрос прерывания приходит в некоторый момент работы прерываемой программы. Как бы ни была организована система прерываний в ЭВМ, между поступлением запроса прерывания и началом выполнения программы-обработчика прерывания неизбежно проходит некоторое время $t_р$, называемое **временем реакции** системы прерываний. Время реакции определяется для запроса прерывания, имеющего наивысший приоритет. Прежде чем обработчик прерывания приступит непосредственно к обработке, он должен обеспечить полноценное восстановление прерываемой программы.

Общее время $t_о$ выполнения программы-обработчика прерывания складывается из следующих составляющих:

- времени $t_с$ сохранения слова состояния прерываемой программы, другой информации, необходимой для ее продолжения, а также тех регистров, значения которых будут изменяться программой-обработчиком прерывания;
- времени $t_п$ исполнения так называемых «полезных» команд, т. е. тех команд, с помощью которых непосредственно происходит обработка события, вызвавшего прерывание;
- времени $t_в$ восстановления слова состояния прерываемой программы и прочей сохраненной информации.

Кроме указанных временных характеристик, часто выделяют **время обслуживания прерывания** $t_{обс}$ — разность между полным временем выполнения прерывающей программы ($t_о$) и временем выполнения всех полезных команд, т. е. $t_{обс} = t_с + t_в$.

Под **глубиной прерывания** понимают максимальное число программ, которые могут прерывать друг друга. Возможны случаи:

- только один запрос воспринимается системой;
- глубина прерывания фиксирована (n);
- программы могут сколько угодно раз прерывать друг друга.

Очевидно, что чем больше глубина прерывания, тем с большей эффективностью можно организовать приоритетное обслуживание. Так, если при глубине прерывания n во время обработки n-го запроса пришел $(n + 1)$-й запрос, то он будет принят к исполнению только после обработки n-го прерывания, даже если $(n + 1)$-й запрос имеет наивысший приоритет.

Если время реакции $t_р$ или время обработки запроса $t_о$ настолько велики, что запрос окажется необслуженным к моменту прихода нового запроса того же типа, то возникает явление, называемое **насыщением системы прерывания**. В этом случае очередной запрос того или иного типа может быть утрачен. Параметры системы прерываний должны быть согласованы таким образом, чтобы насыщение системы не наступило.

Рассмотрим последовательность действий системы прерываний по обработке поступившего запроса более подробно.

2.6.3. Определение момента прерывания и приоритета запроса

Прежде чем начать обработку запроса, система прерываний программ должна определить допустимый момент прерывания текущей программы. Наиболее простыми являются три следующих способа определения допустимого момента прерывания.

Метод помеченного оператора (опорных точек). Суть метода состоит в следующем. В специальные разряды команд, после которых допускается прерывание, записывается определенный знак, разрешающий (например, состояние 1 специального бита команды) или запрещающий (например, состояние 0) прерывание. Команды, после выполнения которых разрешается прерывание, называются **опорными точками**. Опорные точки вдоль программы расставляются таким образом, чтобы информация, находящаяся в регистрах процессора после выполнения команды опорной точки, дальше не использовалась. Поскольку отсутствует необходимость хранения дополнительной информации, в момент прерывания достаточно сохранить лишь ССП текущей программы ($t_{обс} \rightarrow \min$). В то же время обработка вновь пришедшего запроса прерывания будет отложена до ближайшей опорной точки прерываемой программы. Поэтому метод опорных точек уменьшает время обслуживания, но увеличивает время реакции.

Покомандный способ. Здесь прерывание допускается после выполнения любой команды. Способ прост в реализации, при этом $t_р$ уменьшается, а $t_{обс}$ увеличивается по сравнению с предыдущим методом, так как обработчик прерывания вынужден сохранять значения тех регистров процессора, которые он будет неизбежно изменять во время своего выполнения.

Метод быстрого реагирования. Прерывание допускается во время выполнения любой команды, т. е. после выполнения очередного ее такта. Здесь $t_р \rightarrow \min$, а $t_{обс} \rightarrow \max$, так как необходимо запоминать и затем восстанавливать некоторые обычно программно-недоступные элементы (например, счетчик тактов и т. п.).

Хотя покомандный способ сравнительно прост в реализации и имеет наиболее удачное сочетание временных характеристик, в современных ЭВМ, как правило, используется метод быстрого реагирования. Причиной является то, что только этот способ обеспечивает эффективную мультипрограммную обработку. Так, если адрес операнда принадлежит внешней памяти, то текущая команда не может быть продолжена, пока в оперативную память не будет считан блок данных из внешней памяти. Обращение к внешней памяти возможно только средствами операционной системы, поэтому в данном случае необходима генерация логического прерывания во время исполнения очередной команды.

Следующей задачей системы прерываний является определение наиболее приоритетного незамаскированного запроса на прерывание.

Понятие приоритета в прерывании программ имеет два смысла. Предположим, что никаких ограничений на время поступления запросов прерывания не накладывается. При одновременном поступлении нескольких запросов для немедленного удовлетворения может быть принят только один. Этот тип приоритета, определяющий очередность рассмотрения запросов прерывания, называют *приоритетом между запросами прерываний*. Прерывающие программы могут иметь относительно текущей программы различную степень важности, и не любым запросом может быть прервана выполняющаяся программа. Чтобы иметь возможность прервать текущую программу, принятый запрос должен соответствовать программе более важной, нежели выполняемая в данный момент. Этот тип приоритета определяет старшинство программ, и его обычно называют *приоритетом между прерывающими программами*.

В случае простейшей аппаратной реализации приоритета между запросами прерывания может быть использована цепочечная однотактная схема (рис. 2.8) («дейзи-цепочка»).

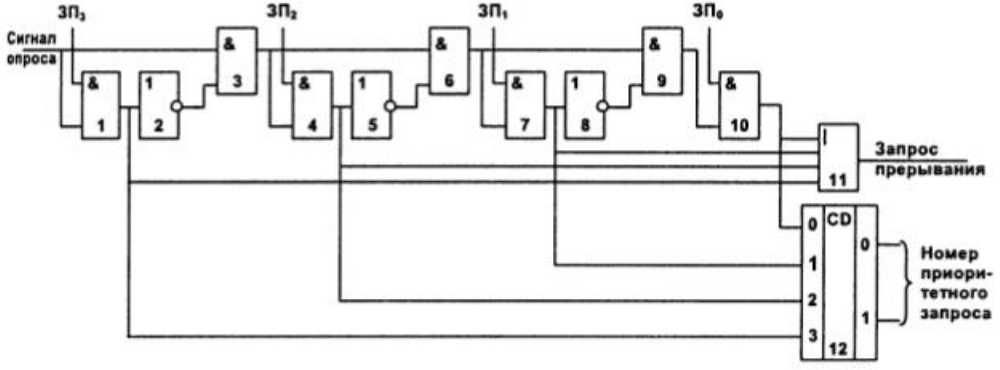


Рис. 2.8. Схема определения номера наиболее приоритетного запроса

Данная схема используется для анализа запросов аппаратных и логических прерываний. Приоритет запросов прерываний ($ЗП_i$) уменьшается с уменьшением номера запроса. В тот момент когда ЭВМ должна определить наличие и приоритет прерывания, процессор выдает сигнал опроса. Если на каком-либо входе $ЗП_i$ присутствует сигнал высокого уровня (есть запрос с номером i), то на элементе 11 (логическое «или») формируется общий сигнал наличия прерывания. Дальнейшее прохождение сигнала опроса блокируется элементами 1–10. Например, если пришел запрос $ЗП_2$, то по сигналу опроса на выходе элемента 4 (логическое «и») появится логическая единица, которая будет подана на вход элементов 11, 12 и 5. Инвертор 5 обратит единицу на входе в ноль на выходе, который с помощью элемента 6 (логическое «и») заблокирует дальнейшее (к элементу 7) прохождение сигнала опроса. На шифраторе (элемент 12) формируется номер поступившего

запроса прерывания, который передается в процессор лишь при наличии общего сигнала запроса прерывания.

Такая структура позволяет быстро анализировать наличие сигнала запроса прерывания и определять наиболее приоритетный запрос из нескольких присутствующих в данный момент. Распределение приоритетов запросов прерываний внешних устройств осуществляется путем их физической коммутации по отношению к процессору. Указание приоритетов — жесткое и не может быть программно изменено. Изменение приоритетов возможно только путем физической перекоммутации устройств.

Приоритет между прерывающими программами определяет, какие из программ могут прервать данную программу, а какие нет. Этот вид приоритета для многоуровневых систем с достаточной глубиной прерывания имеет гораздо большее значение, чем приоритет между запросами прерывания.

Приведем пример. Пусть одновременно возникли три запроса (рис. 2.8) — $ЗП_1$, $ЗП_2$, $ЗП_3$, обработчиками данных запросов являются программы № 1, № 2, № 3 соответственно. По приоритету предпочтение отдается запросу с большим номером. Предположим, что приоритет между прерывающими программами установлен в обратном порядке. Тогда в соответствии с приоритетами между запросами вначале процессор приступает к выполнению программы № 3, но, как только поиск запросов прерывания возобновится, запрос $ЗП_2$ прервет выполнение программы № 3 и запустит программу № 2. Аналогично последняя прервется программой № 1, которая и выполнится первой.

Таким образом, приоритет между запросами прерывания нужен лишь для выбора одного запроса из многих, а приоритет между прерывающими программами определяет фактический порядок выполнения программ.

Так как степень важности программ, их объем, требуемые ресурсы и т. п. могут изменяться в ходе вычислительного процесса, то только приоритеты между запросами прерывания могут быть строго зафиксированы. Приоритеты же между прерывающими программами должны быть программно-управляемыми.

2.6.4. Передача управления прерывающей программы и возврат из нее

Определение начального адреса обработчика прерывания можно осуществлять как программным, так и аппаратным способом.

Суть программного распознавания состоит в следующем. Все линии связи, по которым приходят запросы прерывания, объединяются в схему ИЛИ, формирующую на выходе один и тот же сигнал, который в допустимый момент прерывания поступает в программу-обработчик. Последняя распознает запросы и разветвляется для их выполнения. Так как происходит анализ запросов прерывания,

то времена реакции $t_р$ и обслуживания $t_{обс}$ сравнительно велики, что затрудняет приоритетное обслуживание.

При аппаратном распознавании причин прерывания каждому источнику прерывания ставится в соответствие свой адрес начала программы-обработчика. Специальная схема при появлении запроса формирует соответствующий адрес перехода к прерывающей программе.

Вход системы прерывания, обладающей способностью формировать собственный адрес начала прерывающей программы, принято называть *уровнем прерывания*.

Простейший способ указания начальных адресов состоит в следующем. Каждому уровню присваивается номер и в памяти отводится ячейка, адрес которой, к примеру, равен номеру уровня. В такой ячейке памяти может храниться команда перехода к остальной части обработчика прерывания, которая может находиться в любом месте памяти. Набор фиксированных по отношению к своим уровням ячеек памяти образует таблицу входов в прерывающие программы, содержание которой может менять программист.

Обычно элемент такой таблицы называют *вектором прерывания*, а саму таблицу — *таблицей векторов прерываний* (рис. 2.9). В общем случае вектор прерывания представляет собой слово состояния программы-обработчика. Если каждое ССП занимает в оперативной памяти s байт, то вектор прерывания уровня i будет находиться по адресу $(i-1)\cdot s$, а таблица векторов прерываний с N входами займет первые $N\cdot s$ байт оперативной памяти.

Адрес ОП	Содержимое ячейки ОП
0	**ССП обработчика уровня 0**
s	**ССП обработчика уровня 1**
2 · s	**ССП обработчика уровня 2**
.....	
(N – 1)·s	**ССП обработчика уровня N**

Рис. 2.9. Таблица векторов прерываний с N входами

Заметим, что удобный компромисс достигается от сочетания аппаратного и программного методов распознавания типа запроса прерывания и определения начального адреса обработчика. Можно организовать систему, в которой каждый уровень используется для обслуживания нескольких источников прерывания, а

физически уровень реализуется в виде некоторого количества прерывающих входов, объединенных схемой ИЛИ. В определенную для данного уровня прерывания ячейку памяти записывается состояние прерывающих входов в момент прерывания.

Для осуществления возврата к прерванной программе необходимо полностью восстановить ее состояние вычислений на момент прерывания. Информацию, которую следует сохранять при прерывании программы, можно разделить на основную (которая составляет ССП и запоминается всегда) и дополнительную (необходимость запоминания которой зависит от содержания программы-обработчика).

В слово состояния программы обычно включается:

- содержимое программного счетчика, т. е. адрес первой невыполненной команды прерванной программы;
- триггер состояния системы: «рабочее» или «состояние ожидания»;
- маска прерывания, устанавливаемая каждой новой программой;
- код прерывания — двоичное число, отдельное для каждого уровня, объединяющего прерывание от нескольких источников, по которому прерывающая программа опознает конкретный источник прерывания.

Маска прерывания — шаблонная последовательность знаков, управляющая сохранением или исключением отдельных частей другой последовательности знаков. В простейшем виде маска — это двоичное число, каждый разряд которого соответствует одному из уровней прерывания и разрешает (например, состояние 1) или запрещает (состояние 0) прерывание от запросов, относящихся к данному уровню. Для каждого обработчика прерывания может быть установлена своя маска, указывающая, какие программы-обработчики способны его прерывать. Каждый разряд маски соответствует отдельной программе. Маска прерывания является составной частью слова состояния программы, и если программа вызывается для выполнения, то ее маска засылается в регистр маски. Последний обычно реализуется в виде триггерного регистра, состояние которого можно изменить программным путем. При формировании маски состояние «разрешено» получают лишь те триггеры, которые соответствуют программам с более высоким, чем у данной программы, приоритетом.

Код прерывания обычно находится в общем для всех уровней регистре, т. е. с приходом новой прерывающей программы его также надо запоминать.

В момент прерывания старое ССП, относящееся к прерываемой программе, заменяется ССП программы-обработчика, а в конце прерывания старое ССП восстанавливается прерывающей программой. Для ускорения процесса замены ССП эту процедуру выполняют обычно аппаратным путем. Важным замечанием является то, что в *период сохранения и восстановления ССП прерывания любого уровня запрещены*.

К дополнительной информации относят содержимое:

- арифметических регистров;
- индексных регистров;
- прочих программно-доступных регистров, общих для всех программ, и т. п.

Сохранение дополнительной информации увеличивает время обслуживания. Поэтому программисту надо тщательно продумать, что из дополнительной информации следует запоминать в каждом конкретном случае. Более того, момент прерывания следует выбирать так (если это возможно), чтобы дополнительной информации для сохранения было как можно меньше.

Вопросы для самоконтроля

1. Перечислите основные принципы фон-неймановской модели ЭВМ.
2. Перечислите основные функциональные блоки последовательной ЭВМ.
3. Что собой представляет центральный процессор?
4. Что такое стандартный цикл команды?
5. Поясните разницу между аппаратным и микропрограммным управлением.
6. Изобразите структуру оператора машинной команды.
7. Что такое система команд ЭВМ?
8. Перечислите основные признаки, по которым классифицируются системы команд.
9. Какие принципы заложены при создании архитектур CISC и RISC?
10. Что собой представляет набор команд VLIW?
11. Перечислите основные наборы команд, классифицируемые по месту хранения операндов.
12. Что собой представляет система команд на базе аккумуляторов?
13. Перечислите основные достоинства и недостатки системы команд с использованием РОНов.
14. Приведите примеры машин с архитектурами CISC, RISC, VLIW, а также машин, использующих систему команд на базе стека, аккумуляторов и РОНов.
15. Дайте определение понятию «формат команды».
16. Какие факторы следует учитывать при разработке системы команд?
17. По какому соотношению можно оценить общую длину команды?
18. Сколько и какие форматы команд используются в ЭВМ?
19. Какие факторы учитываются при выборе формата команды?
20. Что такое «способ адресации»?
21. Перечислите основные способы адресации. Отметьте их достоинства и недостатки.

22. Почему в ЭВМ необходима система прерываний? Какие недостатки заложены в модели фон Неймана?

23. Какие характеристики системы прерываний Вы знаете?

24. Перечислите последовательность основных действий системы при возникновении запроса прерывания.

25. В какие моменты времени можно прервать выполняющуюся программу?

26. Чем отличаются приоритеты запросов от приоритетов программ, и как они определяются?

27. Для чего применяется таблица векторов прерываний?

28. Как организуется возврат из обработчика прерывания к выполнению прерванной программы?

29. Что такое насыщение системы прерываний?

ГЛАВА 3. АРХИТЕКТУРА ПАМЯТИ ЭВМ

3.1. ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ПАМЯТИ ЭВМ

В любой ЭВМ, независимо от ее архитектуры, программы и данные хранятся в памяти. Обычно под памятью понимают совокупность устройств, служащих для запоминания, хранения и выдачи информации. Отдельные устройства, входящие в эту совокупность, и называют запоминающими устройствами или памятью того или иного типа. В литературе эти термины зачастую употребляются как синонимы. Однако термин «запоминающее устройство» (ЗУ) обычно употребляют, когда речь идет о принципе построения некоторого устройства памяти (полупроводниковые ЗУ, ЗУ на магнитных лентах, дисках и др.), а термин «память» — когда хотят подчеркнуть выполняемую устройством памяти логическую функцию или место расположения в составе оборудования ЭВМ (оперативная память, внешняя память и т. д.).

Важнейшими характеристиками памяти являются емкость (объем) и быстродействие. Объем памяти характеризуют числом битов или байтов (килобайтов, мегабайтов, гигабайтов и т. д.), которое может храниться в ЗУ. Быстродействие характеризует время, затрачиваемое на обращение к памяти. При оценке быстродействия необходимо учитывать применяемый **метод доступа** к памяти. Различают четыре основных метода доступа.

1. **Последовательный доступ**. Информация в памяти с последовательным доступом хранится в виде последовательности блоков данных, называемых **записями**. Для доступа к нужному блоку (слову, байту) необходимо прочитать все предшествующие ему данные. Хотя пропускная способность ЗУ с последовательным доступом может быть велика, время доступа к памяти определяется главным образом **задержкой**, которая зависит от положения требуемой записи в последовательности записей на носителе. Последовательный доступ применяется в ЗУ на магнитной ленте.

2. **Прямой доступ**. Данные в памяти с прямым доступом также хранятся блоками-записями. Но в отличие от предыдущего метода для обращения к определенной записи нет необходимости считывать предшествующие ей данные, отсюда и название метода — доступ к записи осуществляется «напрямую» по уникальному адресу, отражающему ее физическое размещение в ЗУ. Тем не менее время доступа напрямую зависит от физического расположения блока данных по отношению к текущему положению устройства считывания/записи. После осуществления обращения к определенной записи производится последовательный доступ к единицам информации (словам, байтам) внутри записи. Прямой доступ характерен для ЗУ на магнитных и оптических дисках.

3. **Произвольный доступ**. Память делится на ячейки (слова, байты). Обращение к любой ячейке в отличие от последовательного и прямого доступа за-

нимает одно и то же время и может производиться в произвольной очередности. Произвольный доступ применяется в ЗУ оперативной памяти.

4. **Ассоциативный доступ**. В отличие от предыдущих методов, где обращение к элементам данных происходит по уникальному адресу, ассоциативный доступ осуществляется по содержимому элементов данных. Требуемое содержимое задается с помощью специального образца (шаблона, маски), сравнение с которым осуществляется для всех ячеек памяти, как правило, параллельно. Дальнейшее обращение (чтение, запись) производится только к тем ячейкам, сравнение содержимого которых с образцом дало положительный результат. Память, построенная по подобному принципу, получила название ассоциативной. Ассоциативный доступ часто применяется в кэш-памяти.

Для количественной оценки быстродействия памяти обычно используют три параметра:

1. **Время доступа (T_A)**. Соответствует времени от момента выдачи запроса на чтение/запись данных до момента, когда данные действительно начнут считываться или записываться в память. Время доступа также еще называют **задержкой**, или **латентностью**. Например, при чтении из оперативной памяти время доступа — это время между поступлением адреса на адресную шину и появлением данных на шине данных.

2. **Время передачи данных (T_T)**. Соответствует времени от момента начала непосредственной передачи данных до момента ее завершения. Полностью определяется **скоростью передачи** ***R***, измеряемой в битах (байтах, мегабайтах) в секунду.

3. **Длительность *(время)* цикла обращения**, или **период обращения (T_P)**. Эта величина, которую часто называют просто циклом памяти, означает минимальное время между двумя последовательными обращениями к памяти. Период обращения включает время доступа, время передачи данных плюс некоторое **дополнительное время** T_E, которое требуется для завершения обращения к памяти. В ЗУ оперативной памяти, например, дополнительное время требуется для затухания сигналов на шинных линиях, в ЗУ, где считывание информации приводит к ее разрушению, — для восстановления считанной информации. Заметим, что часто за один цикл обращения можно осуществить чтение или запись лишь ограниченной порции данных длиной *N* байт. В этом случае период обращения вычисляется как

$$T_P = T_A + \frac{N}{R} + T_E. \qquad (3.1)$$

Следующей характеристикой памяти является ее **физический тип**, по которому различают полупроводниковую память, память на магнитных носителях, память на оптических носителях. В историю вычислительной техники отошли память на электронных лампах и ферромагнитная память.

Другой важной характеристикой памяти является **энергозависимость (независимость)**. В энергозависимой памяти информация может быть искажена или потеряна при отключении питающего напряжения. В энергонезависимых ЗУ информация сохраняется и при отключении источника питания. Помимо энергозависимости следует учитывать, приводит ли считывание информации к ее разрушению.

Кроме рассмотренных, память характеризуется таким важным параметром, как **стоимость**, которую принято оценивать отношением общей стоимости ЗУ к его емкости в байтах, т. е. стоимостью хранения одного байта информации.

Всю память ЭВМ можно разделить на **основную (оперативную)** и **внешнюю**.

Главным параметром основной (оперативной) памяти (ОП) является доступное **адресное пространство**. Оно определяется принятой в ЭВМ разрядностью адреса. Если, например, разрядная сетка равна 24 разрядам, то максимальная емкость доступной памяти равна 16 Мбайт, если 32 разряда — то 4 Гбайт (2^{24}–1 и 2^{32}–1 соответственно).

С точки зрения адресации различают несколько типов ОП: *физическая*, *реальная*, *виртуальная*.

Под **физической** ОП понимают конкретное физическое устройство памяти, адресация в котором осуществляется в соответствии с используемой схемой дешифрации адреса и управления выборкой данных.

Под **реальной** ОП понимают устройство ОП, для которого может быть установлено соответствие между адресами обращения и выборкой информации. Объем реальной ОП не превышает объема физической памяти.

Виртуальная ОП (ВП) — это память, границы которой определяются допустимым адресным пространством. Объем ВП может значительно превышать объем реальной ОП.

Как уже отмечалось, в ЭВМ и ВС используют и *внешнюю* память (ВЗУ), реализуемую на различных физических устройствах: магнитных дисках и лентах, оптических дисках и т. п.

3.2. ИЕРАРХИЧЕСКАЯ ОРГАНИЗАЦИЯ ПАМЯТИ ЭВМ

Характеристики объема памяти и ее быстродействия являются взаимно противоречивыми с точки зрения стоимости. Оперативная память минимум на три порядка быстрее внешней памяти, но, как правило, является энергозависимой и стоит значительно дороже и главное — имеет ограниченный объем. Для решения этой проблемы память в ЭВМ организуется по **иерархическому принципу**.

Память современных ЭВМ представляет собой многоуровневую иерархическую структуру, причем более высокий уровень меньше по объему, быстрее и имеет большую стоимость в пересчете на байты, чем более низкий уровень. Уровни иерархии взаимосвязаны: все данные на одном уровне могут быть также найдены

на более 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.

Иерархия памяти обычно состоит из многих уровней, но в каждый момент времени пользуются только два близлежащих уровня. Минимальная единица информации, которая может либо присутствовать, либо отсутствовать в двухуровневой иерархии, называется **блоком**. Размер блока может быть либо фиксированным, либо переменным. Если этот размер зафиксирован, то объем памяти является кратным размеру блока.

Успешное или неуспешное обращение к более высокому уровню памяти называются 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 (hit rate)**, или **коэффициент попаданий (hit ratio)**, есть доля обращений, найденных на более высоком уровне. Иногда она представляется процентами. **Доля промахов (miss rate)** есть доля обращений, которые не найдены на более высоком уровне.

Поскольку увеличение производительности является главной причиной появления иерархии памяти, частота попаданий и промахов является важной характеристикой.

Время обращения при попадании **(hit time)** есть время обращения к более высокому уровню иерархии, которое включает в себя, в частности, и время, необходимое для определения того, является ли обращение попаданием или промахом.

Потери на промах (miss penalty) есть время для замещения блока в более высоком уровне на блок из более низкого уровня плюс время для пересылки этого блока в требуемое устройство. Потери на промах далее включают в себя две компоненты: **время доступа (access time)** — время обращения к первому слову блока при промахе, и **время пересылки (transfer time)** — дополнительное время для пересылки оставшихся слов блока. Время доступа связано с задержкой памяти более низкого уровня, в то время как время пересылки связано с полосой пропускания канала между устройствами памяти двух смежных уровней. Очевидно, что при низком коэффициенте попаданий быстродействие иерархической структуры будет стремиться к быстродействию самого низкого, медленного уровня иерархии.

В современных ЭВМ коэффициент попаданий достаточно высок, что обеспечивается соблюдением **принципа локальности по обращению**. Если рассмотреть процесс выполнения большинства программ, то можно заметить, что с очень высокой вероятностью адрес очередной команды программы либо следует непосредственно за адресом, по которому была считана текущая команда,

либо расположен вблизи него. Такое расположение адресов называется **пространственной локальностью программы**.

В свою очередь, обрабатываемые данные, как правило, структурированы, и такие структуры обычно хранятся в последовательных ячейках памяти. Данная особенность называется **пространственной локальностью данных**. Кроме того, программы содержат множество небольших циклов и подпрограмм, при этом небольшие, локально расположенные наборы команд могут многократно повторяться в течение некоторого интервала времени. Данная особенность называется **временной локальностью программы**.

Все три вида локальности и объединяет понятие **локальности по обращению**. Часто принцип локальности представляют в виде так называемого правила «90/10», означающего, что 90% времени работы программы связано с доступом к 10% адресного пространства этой программы.

Рис. 3.1. Иерархическое построение памяти

Из рис. 3.1 видно, что иерархическое построение памяти дает увеличение емкости памяти. При этом нетрудно увидеть, что обмен данными между быстро-

действующей и медленнодействующей памятью должен осуществляться в соответствии с алгоритмом доступа к каждому ЗУ и является большой нагрузкой при программировании. Как выход из сложившейся ситуации явилось рождение идеи виртуальной памяти. Под **виртуализацией** понимается метод автоматического управления иерархической памятью таким образом, что программисту кажется, что существует одна быстродействующая память большой емкости. На рис. 3.1 показано, как определяется взаимодействие между различными уровнями иерархии памяти.

Самый быстрый, но и минимальный по емкости тип памяти — это внутренние регистры процессора, которые иногда объединяют понятием **сверхоперативное запоминающее устройство — СОЗУ**. Количество внутренних регистров сравнительно невелико, даже в RISC-процессорах не превышает нескольких сотен. Оперативная память значительно большей емкости располагается несколькими уровнями ниже. Между ОП и регистрами размещаются несколько уровней кэш-памяти, которая по емкости существенно проигрывает ОП, но значительно превосходит ее по быстродействию, уступая в то же время регистрам. Уровни кэш-памяти обозначаются буквой L и номером уровня. Современные ЭВМ, как правило, имеют два уровня кэш-памяти (см. рис. 3.1), хотя в новейших разработках появляется также третий уровень (L3), причем разработчики говорят о целесообразности введения и четвертого уровня L4. СОЗУ, кэш-память и ОП образуют *внутреннюю память* ЭВМ.

Обмен информацией между магнитными дисками и основной памятью значительно убыстряется за счет введения дополнительного уровня — дисковой кэш-памяти. Дисковая кэш-память реализуется в виде самостоятельного ЗУ, включаемого в состав магнитного диска.

Нижние уровни иерархии — память на магнитных лентах и оптических дисках — служат для долговременного хранения информации (программ и данных).

В целом можно сказать, что иерархическое построение памяти характеризуется следующими положениями.

1. Имеется несколько иерархических уровней хранения организованной в блоки информации.

2. Иерархические уровни памяти отличаются по быстродействию и емкости, причем более быстродействующие памяти имеют меньшую емкость и располагаются на более высоком уровне иерархии, ближе к процессору.

3. Первое обращение к блоку информации приводит, как правило, к перемещению блока с более медленного уровня иерархии на более быстрый. За счет принципа локальности по обращению, последующие обращения к этому блоку приводят к выборке только из быстродействующей памяти.

Иерархической памяти свойственно уменьшение среднего времени обслуживания ЦП. В некоторых случаях это время характеризуют **эффективным временем цикла обращения** $T_{эф}$. Эта обобщенная характеристика производительности

иерархической памяти является функцией всех компонентов памяти, а также способа взаимодействия между этими компонентами и процессором (или процессорами для случая параллельной ВС).

Если, например, память имеет двухуровневую структуру — ОП и кэш-память, то $T_{эф}$ можно записать в виде следующей функции:

$$T_{эф} = f(T_{ОП}, T_{кэш}, P, C), \qquad (3.2)$$

где $T_{ОП}$ — время цикла ОП; $T_{кэш}$ — время цикла кэш-памяти; P — вероятность промаха; C — некоторая характеристика структурной организации двухуровневой памяти, которая, как правило, определяется экспериментально.

В общем случае для оптимизации структуры управления памятью необходимо:

1) уменьшить вероятность промаха;

2) уменьшить время выборки информации из памяти более высокого уровня;

3) сократить время пересылки информации из памяти более низкого уровня в память более высокого уровня и обратно (сохранение измененной информации);

4) обеспечить целостность данных на всех уровнях иерархии, в том числе и для случая многопроцессорных ВС.

3.3. АРХИТЕКТУРА ОСНОВНОЙ ПАМЯТИ

3.3.1. Принципы построения оперативных запоминающих устройств

Основная память представляет собой единственный, за исключением внутренних регистров, вид памяти, к которой ЦП может обращаться непосредственно. Информация, хранящаяся на внешних ЗУ, становится доступной процессору только после того, как будет переписана в основную память.

ЗУ основной памяти образуют массив ячеек, каждая из которых содержит фиксированное число запоминающих элементов и имеет уникальный адрес, по которому ячейки различаются при обращении для чтения или записи. Обращение к любой ячейке занимает одно и то же время и может производиться в любой последовательности, т. е. ОП образуется из ЗУ с произвольным доступом.

Основная память может включать в себя устройства двух типов: **оперативные запоминающие устройства (ОЗУ)** и **постоянные запоминающие устройства (ПЗУ)**.

ОЗУ, образующие преимущественную долю ОП, допускают как считывание, так и изменение информации. Операции чтения и записи выполняются однотипно, практически с одной и той же скоростью, и производятся с помощью электрических сигналов. В англоязычной литературе аббревиатура ОЗУ обозначается как **RAM** (Random Access Memory — «память с произвольным доступом»). Большинство современных ОЗУ являются энергозависимыми и должны быть постоянно подключены к источнику питания, в противном случае хранимая информация теряется.

Меньшую долю ОП образуют энергонезависимые ПЗУ, обозначаемые **ROM** (Read-Only Memory — «память только для чтения»). Как следует из англоязычного названия, ПЗУ обеспечивают считывание информации, но не допускают ее изменения. Заметим, что в ряде случаев информация в ПЗУ может быть изменена, но этот процесс сильно отличается от считывания и требует значительно большего времени.

ОП современных ЭВМ и ВС реализуется на микросхемах **статических** (Static RAM — **SRAM**) и **динамических** (Dynamic RAM — **DRAM**) ОЗУ. Микросхемы статических ОЗУ имеют меньшее время доступа и не требуют циклов регенерации. Микросхемы динамических ОЗУ характеризуются большей емкостью и меньшей стоимостью, но требуют схем регенерации и имеют значительно большее время доступа.

Обычно микросхемы ОЗУ организуются в виде матрицы ячеек, каждая из которых состоит из одного или более запоминающих элементов (ЗЭ) и имеет свой адрес. Каждый ЗЭ способен хранить один бит информации. Для ЗЭ любой полупроводниковой памяти характерны следующие свойства:

- два стабильных состояния, представляющие двоичные 0 и 1;
- в ЗЭ (хотя бы однажды) может быть произведена запись информации посредством перевода его в одно из двух возможных состояний;
- для определения текущего состояния ЗЭ его содержимое может быть считано.

Моделью ЗЭ динамической памяти служит конденсатор. Бит информации такой памяти представляется в виде наличия (1) или отсутствия (0) заряда на конденсаторе. Считать значение, не изменив эту информацию, невозможно, поэтому после операции считывания заряд ЗЭ необходимо восстановить. Для экономии числа контактов ячейки памяти организованы как элементы некой таблицы. При считывании или записи содержимое целой строки переносится в специальный буфер, а после считывания содержимое буфера перезаписывается в ту же строку ЗЭ динамической памяти, т. е. производится перезаряд конденсаторов.

Время хранения заряда конденсатором из-за паразитных утечек заряда ограничено. Чтобы не потерять имеющиеся данные, необходима периодическая перезапись информации, которая выполняется во время **регенерации** (refresh cycle). Операции разрядки-перезарядки занимают определенное время (в современных ЭВМ — от 1 до 5% от общего времени работы с памятью), снижая скорость работы динамической памяти, что является одним из основных ее недостатков. Однако учитывая информационную емкость, низкую стоимость и энергопотребление, этот тип памяти во многих случаях предпочтительнее статической.

При **матричной организации** ОЗУ (рис. 3.2) реализуется *координатный принцип адресации ячеек*. Адрес ячейки, поступающий по шине адреса ВМ, пропускается через логику выбора, где он разделяется на две составляющие: адрес строки и адрес столбца. Адреса строки и столбца запоминаются соответственно в

регистре адреса строки и регистре адреса столбца микросхемы. Регистры соединены каждый со своим дешифратором. Выходы дешифраторов образуют систему горизонтальных и вертикальных линий, к которым подсоединены запоминающие элементы матрицы, при этом каждый ЗЭ расположен на пересечении одной горизонтальной и одной вертикальной линии.

ЗЭ, объединенные общим «горизонтальным» проводом, принято называть *строкой* (row). Запоминающие элементы, подключенные к общему «вертикальному» проводу, называются *столбцом* (column). Фактически «вертикальных» проводов в микросхеме должно быть, по крайней мере, вдвое больше, чем этого требуется для адресации, поскольку к каждому ЗЭ необходимо подключить линию, по которой будет передаваться считанная и записываемая информация.

Совокупность запоминающих элементов и логических схем, связанных с выбором строк и столбцов, называют *ядром микросхемы памяти*. Помимо ядра, в микросхеме ОЗУ имеется еще интерфейсная логика, обеспечивающая взаимодействие ядра с внешним миром. В ее задачи, в частности, входит коммутация нужного столбца на выход при считывании и на вход — при записи.

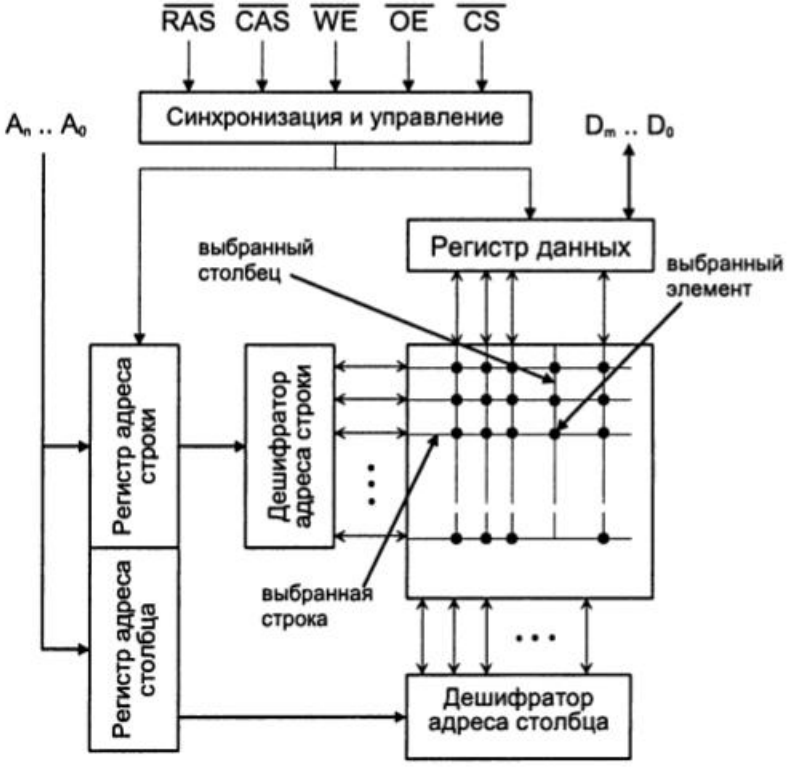


Рис. 3.2. Матричная организация ОЗУ

На физическую организацию ядра, как матрицы однобитовых ЗЭ, накладывается логическая организация памяти, под которой понимается разрядность микросхемы, т. е. количество линий ввода-вывода. Разрядность микросхемы определяет количество ЗЭ, имеющих один и тот же адрес (такая совокупность запоминающих элементов называется **ячейкой**), т. е. каждый столбец содержит столько разрядов, сколько есть линий ввода-вывода данных.

Для сокращения наполовину количества контактов корпуса микросхемы, необходимых для передачи адреса, адресные линии мультиплексируются, адреса строки и столбца в большинстве микросхем подаются в микросхему через одни и те же контакты последовательно во времени и запоминаются соответственно в регистрах адреса строки и столбца микросхемы. Обращение к микросхеме ОЗУ обычно происходит в два этапа. Первый этап начинается с выдачи сигнала **RAS** (Row-Access Strobe — «строб адреса строки»), который фиксирует в микросхеме поступивший адрес строки. Второй этап включает переключение адреса для указания адреса столбца и подачу сигнала **CAS** (Column-Access Strobe — «строб адреса столбца»), который фиксирует этот адрес и разрешает работу регистра данных микросхемы.

Сигнал выбора микросхемы **CS** (Crystal Select) активизирует микросхему и используется для ее выбора в системах, состоящих из нескольких микросхем. Вход **WE** (Write Enable — «разрешение записи») определяет вид выполняемой операции (считывание или запись).

Записываемая информация, поступающая по шине данных, первоначально заносится во входной регистр данных, а затем — в выбранную ячейку. При выполнении операции чтения информация из ячейки до ее выдачи на шину данных буферизуется в выходном регистре данных. Обычно роль входного и выходного выполняет один и тот же регистр. На все время, пока ОЗУ не использует шину данных, информационные выходы микросхемы переводятся в третье (высокоимпедансное) состояние. Управление переключением в третье состояние обеспечивается сигналом **OE** (Output Enable — «разрешение выдачи выходных сигналов»). Этот сигнал активизируется при выполнении операции чтения.

Управление операциями с основной памятью осуществляется контроллером памяти. Обычно этот контроллер входит в состав центрального процессора либо реализуется в виде внешнего по отношению к памяти устройства. Хотя работа микросхем ОЗУ может быть организована как по синхронной, так и по асинхронной схеме, контроллер памяти — устройство синхронное, т. е. срабатывающее исключительно по тактовым импульсам. По этой причине операции с памятью принято описывать с привязкой к тактам. В общем случае на каждую такую операцию требуется, как минимум, пять тактов, которые используются следующим образом:

1) указание типа операции (чтение или запись) и установка адреса строки;
2) формирование сигнала RAS;
3) установка адреса столбца;

4) формирование сигнала CAS;

5) возврат сигналов RAS и CAS в неактивное состояние.

Следует учитывать также задержки, необходимые для стабилизации электрических процессов, порождаемых управляющими сигналами.

Для микросхем динамических ОЗУ дополнительным требованием является необходимость периодической регенерации их состояния. При этом все биты в строке могут регенерироваться одновременно, например путем чтения этой строки. Поэтому ко всем строкам всех микросхем динамических ОЗУ должны производиться периодические обращения в пределах определенного временного интервала порядка 8 миллисекунд.

Это требование, кроме всего прочего, означает, что система основной памяти ЭВМ оказывается иногда недоступной процессору, так как она вынуждена «рассылать» сигналы регенерации каждой микросхеме. Обычно время, затрачиваемое на регенерацию, находится на уровне от 1 до 5% общего времени, а контроллеры памяти включают в свой состав аппаратуру для периодической регенерации динамических ОЗУ.

В отличие от динамических статические ОЗУ не требуют регенерации и имеют значительно меньшее время цикла памяти. Для микросхем, использующих примерно одну и ту же технологию, емкость динамических ОЗУ по грубым оценкам в несколько (4–8) раз превышает емкость статических ОЗУ, но последние имеют в 5–15 раз меньшую длительность цикла и большую стоимость. По этим причинам в основной памяти практически любой ЭВМ или ВС после 1975 г. используются полупроводниковые динамические микросхемы (для построения кэш-памяти при этом применяются статические ОЗУ). Естественно, были и исключения, например в оперативной памяти некоторых моделей супер-ЭВМ компании Cray Research использовались исключительно статические ОЗУ.

3.3.2. Методы повышения быстродействия основной памяти

Задача повышения быстродействия ОП на сегодняшний день является одной из наиболее приоритетных в области построения СВТ. Как уже отмечалось, ОП значительно уступает ЦП в быстродействии и во многих случаях является «узким горлом» всей ЭВМ или ВС. Повышения быстродействия ОП можно добиться за счет применения следующих методов:

1. **Повышение быстродействия ядра микросхем ОЗУ**. Данный метод целиком и полностью опирается на достижения современной микроэлектроники и применяется производителями в виде внедрения все более совершенных технологий производства микросхем. За счет этого удается повысить тактовые частоты работы микросхем памяти.

Разработчики микросхем ОЗУ тратят значительные усилия на повышение их быстродействия, которое принято характеризовать четырьмя параметрами:

- t_{RAS} — минимальное время от перепада сигнала RAS с высокого уровня к низкому до момента появления и стабилизации считанных данных на выходе микросхемы. Среди приводившихся в начале главы характеристик быстродействия это соответствует *времени доступа* T_A;
- минимальное время от начала доступа к одной строке микросхемы памяти до начала доступа к следующей строке. Этот параметр также упоминался в начале главы как *длительность цикла памяти* T_p;
- t_{CAS} — минимальное время от перепада сигнала CAS с высокого уровня к низкому до момента появления и стабилизации считанных данных на выходе микросхемы ($t_{CAS} \approx ¼\ t_{RAS}$);
- T_{PC} — минимальное время от начала доступа к одному столбцу микросхемы памяти до начала доступа к следующему столбцу ($T_{PC} \approx ⅔\ t_{RAS}$).

Возможности «ускорения» ядра микросхемы ЗУ весьма ограничены и связаны в основном с миниатюризацией запоминающих элементов. Очевидно, что, как и повышение тактовых частот микропроцессоров, указанный метод является экстенсивным, хотя и чрезвычайно важным.

2. **Оптимизация доступа к ОЗУ**. Данный метод связан с применением различных *режимов* доступа к ОЗУ. В каждом из режимов в той или иной степени используется свойство локальности по обращению, что позволяет уменьшить среднее время цикла ОЗУ.

3. **Структурные методы повышения быстродействия**. К данным методам относятся блочная организация ОП с чередованием адресов, а также пакетная и конвейерная обработка множества доступов к ОП.

3.3.2.1. Оптимизация доступа к ОЗУ

В данной области достигнуты большие успехи по сравнению с методами повышения быстродействия ядра микросхем ОЗУ. Оптимизация доступа к ОЗУ относится к интерфейсной части микросхем памяти. Методы оптимизации касаются главным образом операции чтения, т.е. способов доставки содержимого ячейки на шину данных. Рассмотрим наиболее распространенные режимы чтения:

- последовательный;
- конвейерный;
- блочный;
- страничный;
- быстрого страничного доступа;
- групповой (пакетный);
- удвоенной скорости;
- синхронный.

Последовательный режим.

При использовании *последовательного режима* (flow through mode) адрес и управляющие сигналы подаются на микросхему до поступления синхроимпульса.

В момент прихода синхроимпульса вся входная информация запоминается во внутренних регистрах — по его переднему фронту, и начинается цикл чтения. Через некоторое время, но в пределах того же цикла, данные появляются на внешней шине, причем момент этот определяется только моментом прихода синхронизирующего импульса и скоростью внутренних цепей микросхемы. Последовательный режим фактически не содержит элементов оптимизации, и, как правило, микросхемы памяти переходят в последовательный режим при так называемых «наихудших» условиях, когда иные режимы работы неприменимы.

Конвейерный режим.

Конвейерный режим (pipelined mode) — это такой метод доступа к данным, при котором можно продолжать операцию чтения по предыдущему адресу в процессе запроса по следующему.

При чтении из памяти время, требуемое для извлечения данных из ячейки, можно условно разбить на два интервала: первый — непосредственно доступ к массиву запоминающих элементов и извлечение данных из ячейки; второй — передача данных на выход (при этом происходит детектирование состояния ячейки, усиление сигнала и другие операции, необходимые для считывания информации). В отличие от последовательного режима, где следующий цикл чтения начинается только по окончании предыдущего, в конвейерном режиме процесс разбивается на два этапа. Пока данные из предыдущего цикла чтения передаются на внешнюю шину, происходит запрос на следующую операцию чтения. Таким образом, два цикла чтения перекрываются во времени. Из-за усложнения схемы передачи данных на внешнюю шину время считывания увеличивается на один такт, и данные поступают на выход только в следующем такте, но такое запаздывание наблюдается лишь при первом чтении в последовательности операции считывания из памяти. Все последующие данные поступают на выход друг за другом, хотя и с запаздыванием на один такт относительно запроса на чтение. Так как циклы чтения перекрываются, микросхемы с конвейерным режимом могут использоваться при частотах шины, вдвое превышающих допустимую для микросхем с последовательным режимом чтения.

Блочный режим.

В *блочном режиме* (nibble mode) ОЗУ может обеспечить выдачу нескольких последовательных ячеек для каждого сигнала RAS. В этом случае за один раз читаются значения m ячеек микросхемы, и они выдаются в выходные регистры в течение четырех оптимизированных циклов. Ускорение до m раз достигается при последовательном чтении соседних ячеек памяти, при этом несколько усложняется схема управления синхросигналами.

Страничный режим.

В основе идеи лежит тот факт, что при доступе к ячейкам со смежными адресами (согласно принципу локальности по обращению такая ситуация наиболее ве-

роятна), причем к таким, где все ЗЭ расположены в одной строке матрицы, доступ ко второй и последующим ячейкам можно производить существенно быстрее. Действительно, если адрес строки при очередном обращении остался прежним, то все временные затраты, связанные с повторным занесением адреса строки в соответствующий регистр микросхемы, дешифровкой, зарядом паразитной емкости горизонтальной линии и т. п., можно исключить. Для доступа к очередной ячейке достаточно подавать на вход микросхемы лишь адрес нового столбца, сопровождая его сигналом CAS. Отметим, что обращение к первой ячейке в последовательности производится стандартным образом — поочередным заданием адреса строки и адреса столбца, т. е. здесь время доступа уменьшить практически невозможно. Рассмотренный режим называется *режимом страничного доступа* или просто *страничным режимом* (page mode). Под страницей понимается строка матрицы ЗЭ. Микросхемы, где используется страничный режим и его модификации, принято характеризовать формулой $x - y - y - y$. Первое число x представляет количество тактов системной шины, необходимое для доступа к первой ячейке последовательности, а y — к каждой из последующих ячеек. Так, выражение 7 – 3 – 3 – 3 означает, что для обработки первого слова необходимо 7 тактовых периодов системной шины (в течение шести из которых шина простаивает в ожидании), а для обработки последующих слов — по 3 периода, из которых два системная шина также простаивает.

Режим быстрого страничного доступа.

Режим быстрого страничного доступа (FPM — Fast Page Mode) представляет собой модификацию стандартного страничного режима. Основное отличие заключается в способе занесения новой информации в регистр адреса столбца. Полный адрес (строки и столбца) передается только при первом обращении к строке. Активизация буферного регистра адреса столбца производится не по сигналу CAS, а по заднему фронту сигнала RAS. Сигнал RAS остается активным на протяжении всего страничного цикла и позволяет заносить в регистр адреса столбца новую информацию не по спадающему фронту CAS, а как только адрес на входе микросхемы стабилизируется, т. е. практически по переднему фронту сигнала CAS. В целом же потери времени сокращаются на два такта, которые ранее требовались для передачи адреса каждой строки и сигнала RAS. Реальный выигрыш, однако, наблюдается лишь при передаче блоков данных, хранящихся в одной и той же строке микросхемы. Если же программа часто обращается к разным областям памяти, переходя с одной строки микросхемы на другую, преимущества метода теряются.

Режим удвоенной скорости.

Важность *режима удвоенной скорости передачи данных* (DDR — Double Data Rate) трудно переоценить. Суть данного метода заключается в передаче данных

по обоим фронтам импульса синхронизации, т. е. дважды за один такт. Тем самым пропускная способность системы памяти увеличивается в два раза.

Групповой (пакетный) режим.

Данный режим (burst mode) является сочетанием блочного и страничного подходов. Разрядность ячейки памяти, как правило, равна одному байту, при этом ширина шины данных составляет не менее четырех байт. Одно обращение к памяти требует последовательного доступа к четырем смежным ячейкам — пакету. При реализации группового режима адрес столбца заносится в микросхему только для первой ячейки пакета, а переход к очередному столбцу производится уже внутри микросхемы. Это позволяет для каждого пакета исключить три из четырех операций занесения адреса столбца и тем самым еще более сократить среднее время доступа. Часто групповой режим сочетается с конвейерным, образуя *пакетно-конвейерный* (pipelined burst mode) режим.

Синхронный режим.

Синхронная (Synchronous DRAM — SDRAM) работает синхронно с тактами центрального процессора. При этом контроллер памяти имеет информацию о времени готовности данных, что позволяет освободить процессор от необходимости ждать момента доступа к памяти. Другими словами, моменты готовности ОЗУ к приему-передаче информации совпадают с моментами обращения процессора к памяти. При обращении к памяти процессор не тратит время на ожидание прихода синхроимпульса микросхемы ОЗУ. Кроме этого, при операциях чтения и записи используется пакетно-конвейерный режим. Благодаря этим приемам скорость работы микросхемы SDRAM почти в четыре раза превышает скорость работы FPM DRAM. Для SDRAM характерна работа по циклу 5 – 1 – 1 – 1 такта для первого и 1 – 1 – 1 – 1 для последующих обращений.

3.3.2.2. Структурные методы повышения быстродействия ОП

Методы данной группы направлены на повышение быстродействия ОП за счет оптимизации структурной организации ОЗУ, состоящего из множества микросхем. Помимо повышения быстродействия ОП, с помощью структурных методов добиваются увеличения разрядности ОП и, как следствие,— повышения ее емкости. К структурным методам относятся:

- блочная организация ОП;
- расслоение ОП (чередование адресов);
- пакетная обработка множества доступов к ОП;
- конвейерная обработка множества доступов к ОП.

Блочная организация ОП.

Как правило, микросхемы ОЗУ (МС ОЗУ) имеют ограниченные емкость и количество электрических входов, т.е. разрядность. Снять эти ограничения можно за счет объединения микросхем ОЗУ в **модули памяти**. Объем памяти моду-

ля получается равным сумме емкостей микросхем ОЗУ, а увеличение разрядности достигается за счет объединения адресных входов объединенных микросхем (рис. 3.3). Информационные входы и выходы микросхем являются входами и выходами модуля памяти увеличенной разрядности. Заметим, что модулем можно считать и единственную микросхему, если она уже имеет нужную разрядность. Один или несколько модулей образуют **банк памяти**.

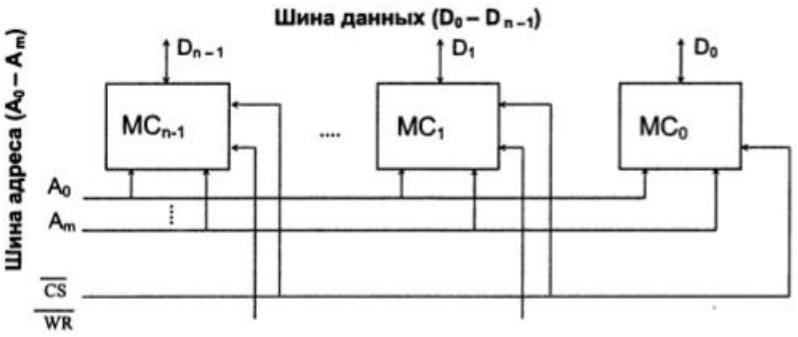


Рис. 3.3. Увеличение разрядности памяти

Для получения требуемой емкости ОП нужно определенным образом объединить несколько банков памяти меньшей емкости. В общем случае основная память ЭВМ практически всегда имеет *блочную структуру*, т. е. содержит несколько банков. При использовании блочной памяти, состоящей из M банков, адрес ячейки A преобразуется в пару (b, w), где b — номер банка, w — адрес ячейки внутри банка.

Могут быть применены три схемы распределения разрядов адреса A между b и w:

- блочная: номер банка b определяется по старшим разрядам адреса A;
- циклическая: $b = A \bmod M$, $w = \dfrac{A}{M}$;
- блочно-циклическая: комбинация двух первых схем.

Пусть необходимо организовать n-разрядную ОП (рис. 3.4), состоящую из $M = 2^m$ банков памяти по 2^{n-m} слов в каждом. Адресное пространство разбивается на группы последовательных адресов, и каждая такая группа обеспечивается отдельным банком памяти. Для обращения к ОП используется n-разрядный адрес, $n - m$ младших разрядов которого (A_{n-m-1} — A_0) поступают параллельно на все банки памяти и выбирают в каждом из них одну ячейку. Старшие m разрядов адреса (A_{n-1} — A_{n-m}) содержат номер банка. Выбор банка обеспечивается либо с помощью дешифратора номера банка памяти, либо путем мультиплексирования информации.

В функциональном отношении такая память может рассматриваться как единое ОЗУ, емкость которого равна суммарной емкости составляющих, а быстро-

действие — быстродействию одного банка. Как видим, блочная организация непосредственно не увеличивает быстродействие ОП, однако без подобной организации невозможно применение остальных структурных методов.

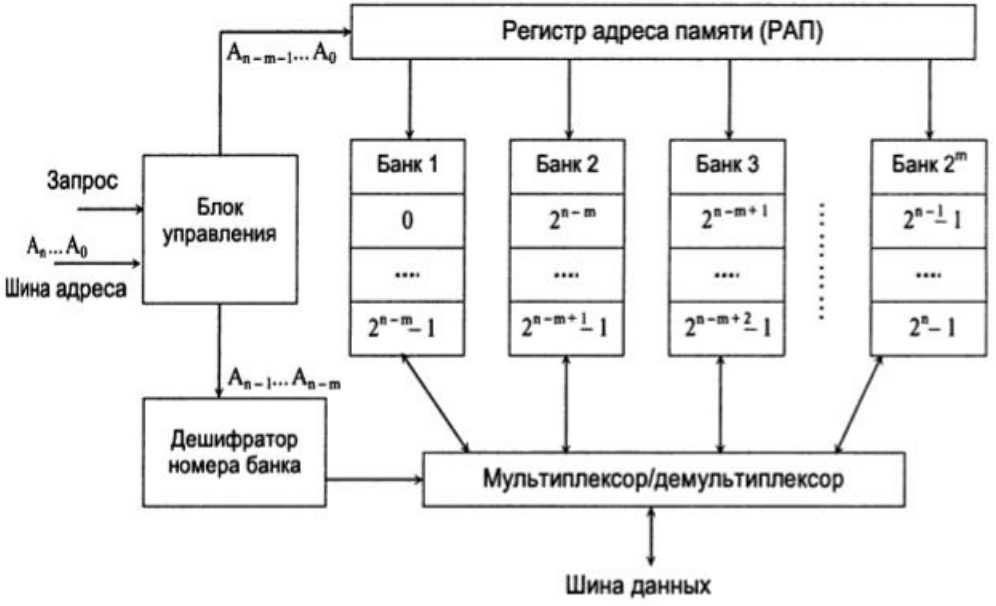


Рис. 3.4. Блочная организация памяти

Расслоение памяти.

Наличие в системе множества банков памяти позволяет использовать потенциальный параллелизм, заложенный в блочной организации. Очевидно, что в каждый момент времени обращение возможно только к одному из слов каждого банка. Чтобы получить большую скорость доступа, нужно осуществлять одновременный доступ ко многим банкам памяти. Одна из общих методик, используемых для этого, называется **расслоением памяти**. В ее основе лежит так называемое **чередование адресов (address interleaving)**, заключающееся в изменении системы распределения адресов между банками памяти.

При расслоении банки памяти обычно упорядочиваются так, чтобы M последовательных адресов памяти A, $A+1$, $A+2$, ..., $A+M-1$ приходились на m различных банков.

Соответствие некоторого адреса ОП A и адреса w внутри банка с номером b определяется соотношением

$$A = \frac{2^n}{M} \cdot b + w, \tag{3.3}$$

где n — разрядность ОП; M — число банков памяти; $\frac{2^n}{M}$ — число слов в одном банке пямяти.

Распределение адресов между M банками памяти называется M-кратным чередованием адресов памяти, соответственно, память, состоящая из M банков с распределенными адресами, называется памятью с M-кратным чередованием адресов. Заметим, что число M является обычно степенью 2.

Используя M-кратное чередование адресов, можно достичь в M раз большей скорости доступа к памяти в целом по сравнению с доступом к отдельному ее банку, если обеспечить при каждом доступе обращение к данным в каждом из банков. Ниже будут рассмотрены способы реализации таких расслоенных структур. Такие системы оптимизируют обращения по последовательным адресам памяти, что является характерным при подкачке информации в кэш-память при чтении, а также при записи в случае использования кэш-памятью механизмов обратного копирования. Однако если требуется доступ к непоследовательно расположенным словам памяти, производительность расслоенной памяти может несколько снижаться.

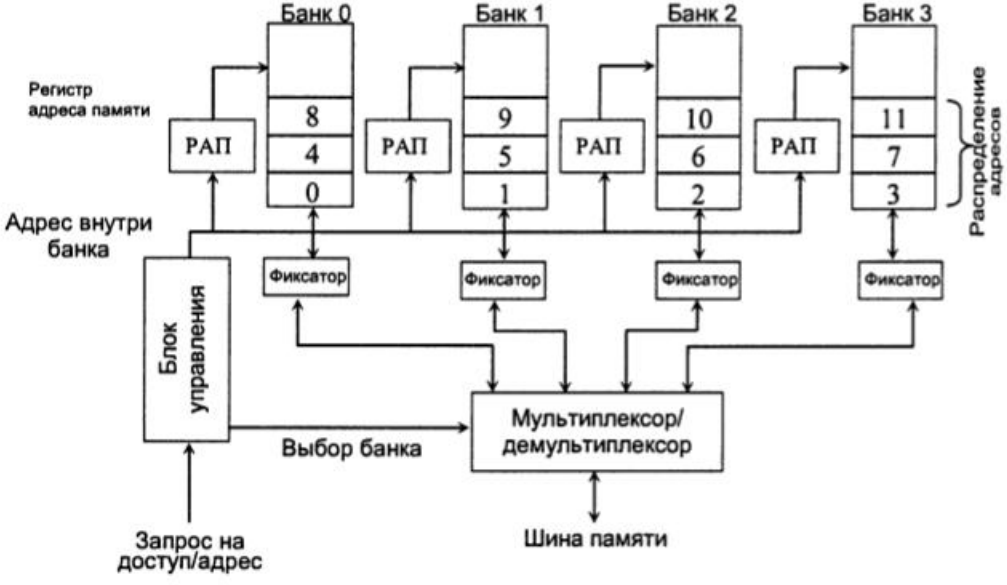


Рис. 3.5. Реализация памяти с 4-кратным чередованием адресов

На рис. 3.5 изображена система с 4-кратным чередованием адресов памяти. Блок управления вычисляет выражение (3.3), определяя по адресу ОП номер банка памяти и адрес слова внутри банка. Последний передается регистрам адреса памяти (РАП) для выборки слова из банка. Из каждого банка памяти слово считывается в регистр-фиксатор данных. Выбор того или иного фиксатора осу-

ществляется блоком управления по номеру банка. Через мультиплексор считанная информация поступает на шину памяти. Запись информации осуществляется аналогично, только в обратном порядке.

Если система памяти разработана для поддержки множества независимых запросов (как это имеет место при работе с кэш-памятью при реализации многопроцессорной и векторной обработки), эффективность системы будет в значительной степени зависеть от частоты поступления независимых запросов к разным банкам. Обращения по последовательным адресам хорошо обрабатываются традиционными схемами расслоенной памяти. Фактически в системе с M-кратным чередованием адресов доступ к памяти возможен в интервале $\frac{1}{M}$ цикла памяти. Однако если в течение M циклов, следующих непрерывно друг за другом, запрашивается доступ к одному и тому же банку, то до окончания предыдущего доступа к этому банку запрос на следующий доступ будет ожидать обработки. Такая ситуация называется **конфликтом по доступу**.

Одно из решений, используемое в больших ЭВМ и ВС для предотвращения конфликтов по доступу, заключается в том, чтобы статистически уменьшить вероятность подобных обращений путем значительного увеличения количества банков памяти. Например, в супер-ЭВМ Cray Y-MP C90 используется 1024-кратное чередование адресов, причем вся память разбита на 8 секций по 8 подсекций по 16 банков памяти в каждой подсекции. Адреса идут с чередованием по каждому из данных параметров. При одновременном обращении к одной и той же секции возникает задержка в 1 такт, а при обращении к одной и той же подсекции одной секции задержка варьируется от 1 до 6 тактов. При выборке последовательно расположенных данных или при выборке с любым нечетным шагом конфликтов по доступу не возникает.

Расслоение памяти обычно реализуется одним из двух способов — с помощью пакетной или конвейерной обработки множества запросов к памяти, хотя возможно и сочетание этих двух способов.

Пакетная обработка множества доступов к памяти.

Данный подход, как и большинство остальных, опирается на принцип локальности по обращению, точнее на одну из его составляющих — принцип пространственной локальности. Команды, как правило (исключение составляют команды переходов), считываются из памяти в строгой последовательности из соседних ячеек памяти. Обмен между ОП и кэш-памятью осуществляется блоками данных, также расположенных в последовательных ячейках. Если сделать возможным одновременный доступ к содержимому определенного числа соседних ячеек памяти, то последовательные запросы на доступ к этим соседним ячейкам можно обрабатывать посредством последовательного доступа к фиксаторам (регистрам, стробам) данных. При этом число последних равно числу запросов. Так как время доступа к фиксаторам данных гораздо меньше времени доступа к памяти, то

среднее время доступа существенно сокращается. На рис. 3.6 представлена схема реализации описанного подхода.

В основе пакетного способа доступа лежит блочная организация памяти с чередованием адресов, так как одновременный доступ к последовательно расположенным словам *a*, *b*, *c* и *d* возможен только при условии, если эти слова размещены в разных банках памяти, и применяется расслоение памяти.

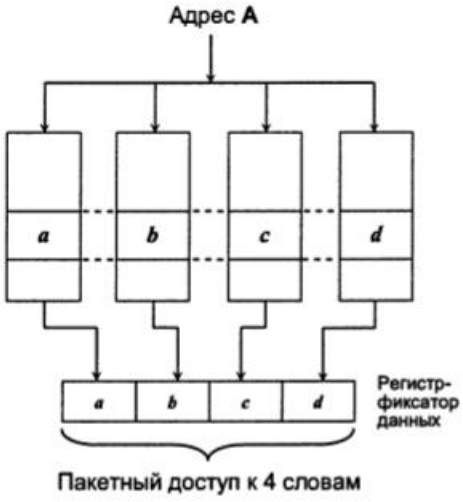


Рис. 3.6. Реализация пакетного способа доступа к памяти

При считывании слова по A-му адресу ОП согласно выражению (3.3) определяется номер банка b. Все значения от адреса $A - b$ до адреса $A + M - b - 1$ считываются в фиксаторы данных. Если существует локальность по обращению, и доступ осуществляется последовательно, начиная с адреса A, к адресам $A + 1$, $A + 2$ и т. д., то считывание данных в промежутке от $A + 1$ до $A + M - b - 1$ можно выполнить без обращения к памяти, а лишь считыванием фиксаторов данных. Поскольку время доступа к фиксатору гораздо меньше доступа к памяти, то среднее время доступа также сокращается.

Конвейерная обработка множества доступов к памяти.

При реализации *конвейерного способа* доступа к памяти ее блочная организация дополняется *очередью запросов* доступа и *очередью данных*. Этот способ позволяет в несколько раз увеличить внутреннее быстродействие памяти по сравнению с внешним быстродействием. Вариант реализации конвейерного способа доступа к памяти представлен на рис. 3.7. Допустим, за время, меньшее времени цикла памяти, поступило некоторое множество запросов к ОП. Поступившие адреса накапливаются в очереди запросов доступа. Сам доступ к каждому адресу из очереди запросов осуществляется последовательно, но не дожидаясь оконча-

ния предыдущего доступа. Считываемые данные помещаются в выходную очередь данных, откуда и поступают в регистр-фиксатор данных.

Рис. 3.7. Реализация конвейерного способа доступа к памяти

Как и в предыдущем случае, конвейерный способ доступа целесообразно применять только при блочной организации памяти, так как существенный выигрыш можно получить только, если слова *a*, *b*, *c* и *d* будут размещены в разных модулях памяти.

Достаточно часто происходит последовательное обращение к адресам A, $A+S$, $A+2S$ и т. д., где S — некоторое целое число (шаг обращения). Такое обращение характерно, например, при векторной обработке данных при доступе к векторным операндам. Обозначим через gcd(M, S) наибольший общий делитель чисел M и S. Когда $\frac{M}{\gcd(M,S)}$ доступов последовательно осуществляются к различным банкам памяти, производительность увеличивается примерно в $\frac{M}{\gcd(M,S)}$ раз по сравнению со случаем доступа к одному банку. Если M и S являются взаимно простыми, то производительность увеличивается в M раз. Если в качестве M (числа банков памяти) выбрать простое число, то множество значений S, взаимно простых с M, увеличится, т. е. повышение производительности будет приближаться к M для более широкого набора значений S. Память с M-кратным чередованием адресов, где M — простое число, называется *первичной памятью*. Примером ВС с первичной памятью является ЭВМ BSP (Burroughs Scientific Processor) фирмы Burroughs.

3.4. ПАМЯТЬ С АССОЦИАТИВНЫМ ДОСТУПОМ

Память с ассоциативным доступом, или **ассоциативная память**, отличается от остальных типов памяти тем, что обращение к ее ячейкам осуществляется не по определенному адресу, а по *содержимому* ячеек памяти. Фактически ассоциативная память работает как поисковая система, способная найти информацию по заданному образцу. Основу ассоциативной памяти составляют **ассоциативные запоминающие устройства (АЗУ)**, которые, как и большинство оперативных ЗУ, являются энергозависимыми и реализуются в виде полупроводниковых микросхем (наборов микросхем).

Принцип работы АЗУ поясняет схема, представленная на рис. 3.8. Запоминающий массив, как и в адресных ЗУ, разделен на m-разрядные ячейки, число которых n. Как правило, в состав АЗУ входят:

- запоминающий массив (ЗМ);
- регистр ассоциативных признаков (РгАП);
- регистр маски (РгМ);
- регистр индикаторов адреса со схемами сравнения на входе.

В АЗУ могут быть и другие элементы, наличие и функции которых определяются способом использования АЗУ.

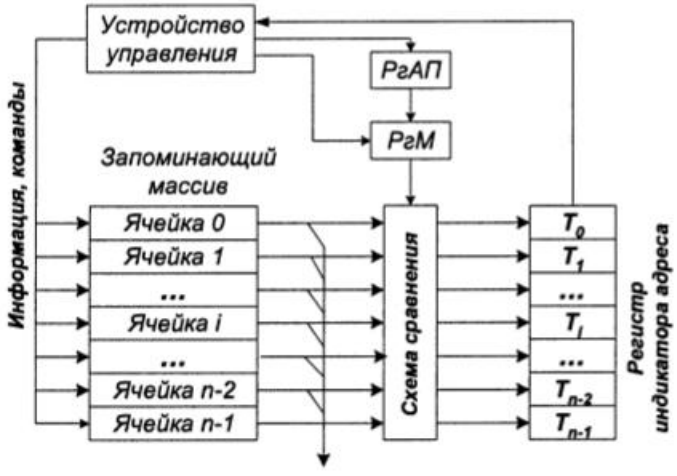


Рис. 3.8. Ассоциативное запоминающее устройство

Выборка информации из АЗУ происходит следующим образом. В регистр ассоциативных признаков из устройства управления передается образец для поиска — **код признака искомой информации** (иногда его называют **компарандом**). Код может иметь произвольное число разрядов — от 1 до m. Если код признаков используется полностью, то он без изменения поступает на схему сравнения, если же необходимо использовать только часть кода, тогда ненужные разряды маскируются с помощью регистра маски. Перед началом поиска информации в АЗУ

все разряды регистра индикаторов адреса устанавливаются в состояние 1. После этого производится опрос первого разряда всех ячеек запоминающего массива и содержимое сравнивается с первым разрядом регистра ассоциативных признаков. Если содержимое первого разряда i-й ячейки не совпадает с содержимым первого разряда РгАП, то соответствующий этой ячейке разряд регистра индикаторов адреса T_i сбрасывается в состояние 0, если совпадает — разряд T_i остается в состоянии 1. Затем эта операция повторяется со вторым, третьим и последующими разрядами до тех пор, пока не будет произведено сравнение со всеми разрядами РгАП. После поразрядного опроса и сравнения в состоянии 1 останутся те разряды регистра индикаторов адреса, которые соответствуют ячейкам, содержащим информацию, совпадающую с записанной в регистре ассоциативных признаков. Эта информация может быть считана в той последовательности, которая определяется устройством управления.

Заметим, что время поиска информации в ЗМ по ассоциативному признаку зависит только от числа разрядов признака и скорости опроса разрядов, но совершенно не зависит от числа ячеек ЗМ. Этим и определяется главное преимущество АЗУ перед адресными ЗУ: в адресных ЗУ при операции поиска необходим перебор всех ячеек запоминающего массива. Кроме того, существуют реализации АЗУ, выполняющих поиск одновременно над всеми разрядами всех слов, записанных в память, т. е. время поиска в подобных устройствах не превышает времени цикла памяти.

Запись новой информации в ЗМ производится без указания номера ячейки. Обычно один из разрядов каждой ячейки используется для указания ее занятости, т. е. если ячейка свободна для записи, то в этом разряде записан 0, а если занята, — 1. Тогда при записи в АЗУ новой информации устанавливается признак 0 в соответствующем разряде регистра ассоциативных признаков, и определяются все ячейки ЗМ, которые свободны для записи. В одну из них устройство управления помещает новую информацию.

Нередко АЗУ строятся таким образом, что, кроме ассоциативной, допускается и прямая адресация данных, что представляет определенные удобства при работе.

Необходимо отметить, что запоминающие элементы АЗУ в отличие от элементов адресуемых ЗУ должны не только хранить информацию, но и выполнять определенные логические функции, поэтому позволяют осуществить поиск не только по равенству содержимого ячейки заданному признаку, но и по другим условиям: содержимое ячейки больше (меньше) компаранда, а также больше или равно (меньше или равно).

Отмеченные выше свойства АЗУ характеризуют преимущества АЗУ для обработки информации. Формирование нескольких потоков идентичной информации с помощью АЗУ осуществляется быстро и просто, а с большим числом операционных элементов можно создавать высокопроизводительные системы. Надо

учитывать еще и то, что на основе ассоциативной памяти легко реализуется изменение места и порядка расположения информации. Благодаря этому АЗУ является эффективным средством формирования наборов данных.

Исследования показывают, что целый ряд задач, таких, как обработка радиолокационной информации, распознавание образов, обработка различных снимков и других задач с матричной структурой данных, эффективно решается ассоциативными системами. К тому же программирование таких задач для ассоциативных систем гораздо проще, чем для традиционных.

К сожалению, устройства памяти с ассоциативным доступом имеют высокие сложность изготовления и стоимость, превышающие аналогичные показатели как динамических, так и статических ОЗУ. Ассоциативная память является основой для построения параллельных ассоциативных систем, а также для ВС, управляемых потоком данных. Наиболее же широко ассоциативный доступ применяется в подсистемах кэш-памяти.

3.5. КЭШ-ПАМЯТЬ

Впервые двухуровневое построение памяти было предложено М. Уилксом в 1965 г. при построении ЭВМ Atlas. Суть подхода заключалась в размещении между ЦП и ОП быстродействующей буферной памяти небольшого размера. В процессе работы ЭВМ те участки ОП, к которым ведется обращение, копируются в буферную память. За счет соблюдения принципа локальности по обращению получается существенный выигрыш в производительности.

Новый вид памяти получил название **кэш-память** (от англ. **cache** — «тайник, убежище»), поскольку такая память скрыта, «невидима» для ЦП, который не может непосредственно обратиться к ней. В свою очередь, программист может вообще не знать о существовании кэш-памяти. В серийных ЭВМ кэш-память впервые была применена в системах модели 85 семейства IBM S/360. Сегодня кэш-память есть в любом классе ЭВМ, причем зачастую имеет многоуровневую структуру.

Все термины, которые были определены раньше, могут быть использованы и для кэш-памяти, хотя слово «**строка**» (**line**) часто употребляется вместо слова «**блок**» (**block**).

Как правило, кэш-память строится на основе сверхбыстродействующих и дорогостоящих ОЗУ статического типа, при этом ее быстродействие в 5–10 раз превышает быстродействие ОП, а объем — в 500–1000 раз меньше. Заметим, что увеличению объема кэш-памяти по отношению к емкости ОП препятствует не только и не столько высокая стоимость статических ОЗУ. Дело в том, что при увеличении емкости кэш-памяти возрастает сложность схем управления, что, в свою очередь, ведет к падению быстродействия. Многочисленные исследования показали, что указанное соотношение объемов кэш-памяти и ОП является оптимальным и будет сохраняться в процессе развития ЭВМ при увеличении быстродействия обоих видов памяти.

Как уже было сказано, ЦП не имеет непосредственного доступа к кэш-памяти. За организацию взаимодействия ЦП, ОП и кэш-памяти отвечает специальный контроллер. Вся ОП разбивается на блоки фиксированного объема, при этом старшая часть адреса ОП определяет **адрес блока**, а младшая часть — **адрес слова внутри блока**. Обмен информации между ОП и кэш-памятью осуществляется блоками. Кэш-память также имеет свою внутреннюю адресацию, и каждый считанный из ОП блок размещается в кэш-памяти по определенному **адресу блока в кэш-памяти**. Часто блоки кэш-памяти называются **строками**, или **кэш-строками**.

Если блок, к которому осуществляется запрос со стороны ЦП, находится в кэш-памяти, то его считывание завершается уже при обращении к кэш-памяти. Таким образом, обеспечивая доступ к некоторому адресу, контроллер должен сначала определить, имеется ли в кэш-памяти копия блока, содержащего этот адрес, и, если имеется, то определить, с какого адреса кэш-памяти начинается этот блок. Эту информацию контроллер получает с помощью **механизма преобразования адресов**. Сложность этого механизма зависит от **стратегии размещения**, определяющей, в какое место кэш-памяти следует поместить каждый блок ОП.

Не менее важным является вопрос о том, в какой момент нужно помещать в кэш-память копию блока из ОП. Данный вопрос решается с помощью **стратегии выборки**.

При записи в кэш-память существует несколько методов замещения старой информации, которые определяются **стратегией обновления основной памяти**.

Часто возникает ситуация, когда, несмотря на выборку из ОП необходимого блока, в кэш-памяти нет места для его размещения. В этом случае необходимо выбрать одну из кэш-строк и заменить ее новым блоком. Способ определения удаляемой кэш-строки называется **стратегией замещения**.

3.5.1. Стратегии размещения

Существует следующие способы размещения данных в кэш-памяти:

- прямое распределение;
- полностью ассоциативное распределение;
- частично (множественно) ассоциативное распределение.

Допустим, разрядность шины адреса n, тогда емкость ОП $V_{оп} = 2^n$ слов. Без ограничения общности определим размер кэш-строки в 256 слов, таким образом, вся ОП будет поделена на 2^{n-8} блоков. В адресе ОП старшие $n-8$ битов будут определять адрес блока, а младший байт— адрес слова в блоке. Пусть емкость кэш-памяти $V_{кэш}$ в 1024 раза меньше емкости ОП, т. е. $V_{кэш} = 2^{n-10}$ слов или 2^{n-18} блоков (кэш-строк).

3.5.1.1. Прямое распределение

Если каждый блок основной памяти имеет только одно фиксированное место, на котором он может появиться в кэш-памяти, то такая кэш-память называется **кэшем с прямым распределением** (direct mapped cache). Это наиболее простая

организация кэш-памяти, при которой для отображения адресов блоков ОП на адреса кэш-памяти просто используются младшие разряды адреса блока. Таким образом, все блоки ОП, имеющие одинаковые младшие разряды в своем адресе, попадают в одну кэш-строку, т. е.

(адрес кэш-строки) = (адрес блока ОП) mod (число блоков в кэш-памяти).

В нашем примере адрес кэш-строки *c* будут составлять младшие $n - 18$ бит адреса блока ОП (рис. 3.9). Преобразование адреса блока ОП в адрес кэш-строки осуществляется путем выборки этих младших $n - 18$ бит. По этому адресу кэш-строки может быть помещен любой из 1024 блоков ОП, имеющих одинаковые $n - 18$ младших бит. Между собой эти блоки будут различаться старшими 10-ю битами *t*, называемыми **тегом**. Для того чтобы определить, какой именно блок ОП хранится в данное время в кэш-памяти, используется еще одна память — так называемая **память тегов (теговая память)**. Теговая память адресуется пословно, причем каждое слово имеет размер, равный размеру тега. Емкость памяти тегов — это произведение размера тега на общее число кэш-строк, для нашего примера составляет $10 \cdot 2^{n-18}$ бит. Адресом памяти тегов является адрес кэш-строки *c*. В отличие от памяти тегов память, в которой хранятся блоки, помещенные в кэш, называется **памятью данных**. Память данных адресуется пословно, ее адрес образуется из адреса кэш-строки и адреса слова внутри блока (кэш-строки).

Рис. 3.9. Структура адреса памяти при прямом распределении

При доступе к *A*-му адресу ОП (рис. 3.10) младшие $n - 18$ бит адреса блока (поле *c*), где содержится этот адрес, используются в качестве адреса кэш-строки. По адресу кэш-строки из теговой памяти считывается тег (поле *t*). Параллельно этому осуществляется доступ к памяти данных с помощью $n - 10$ младших бит адреса *A* (поля *c* и *w*). Если считанный тег и старшие 10 бит адреса *A* совпадают, то это означает, что блок, содержащий адрес *A*, существует в памяти данных, и в слове, к которому осуществляется доступ, хранится копия *A*-го адреса ОП.

Если тег отличается от старших 10 бит адреса *A*, то из основной памяти считывается блок, содержащий адрес *A*, а из кэш-памяти удаляется кэш-строка, чей

адрес определяется полем *c* (младшими *n* – 18 битами) адреса считываемого блока. На место удаленной кэш-строки помещается считанный из ОП блок, при этом обновляется соответствующий тег в памяти тегов.

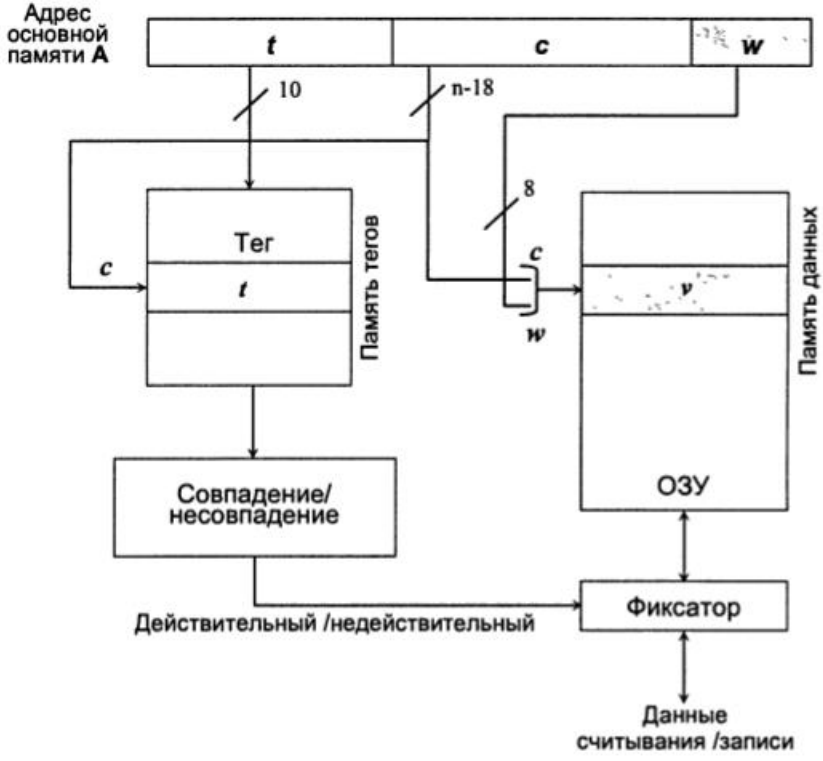


Рис. 3.10. Организация кэш-памяти с прямым распределением

Достоинством прямого распределения является простота реализации, однако из-за того, что адрес кэш-строки однозначно определяется адресом блока ОП, высока вероятность сосредоточения областей блоков в некоторой части кэш-памяти. Замена блоков в этой части будет происходить довольно часто, в то же время другие области кэш-памяти могут простаивать. В такой ситуации эффективность кэш-памяти заметно снижается.

3.5.1.2. Полностью ассоциативное распределение

Если некоторый блок основной памяти может располагаться на любом месте кэш-памяти, то кэш называется **полностью ассоциативным (fully associative)**. Структура адреса основной памяти при таком распределении показана на рис. 3.11.

В этом случае (рис. 3.12) при преобразовании адреса необходимо быстро получить ответ, существует ли копия блока с произвольно указанным адресом в кэш-памяти, и если существует, то по какому адресу. Для осуществления бы-

строго преобразования применяется ассоциативная память тегов. На вход ассоциативной теговой памяти в качестве ключа поиска подается адрес блока *t* разрядностью $n - 8$ бит, являющийся в этом случае тегом, а на выходе получается 10-разрядный адрес кэш-строки *c*, т. е. адрес запрашиваемого блока внутри кэш-памяти данных.

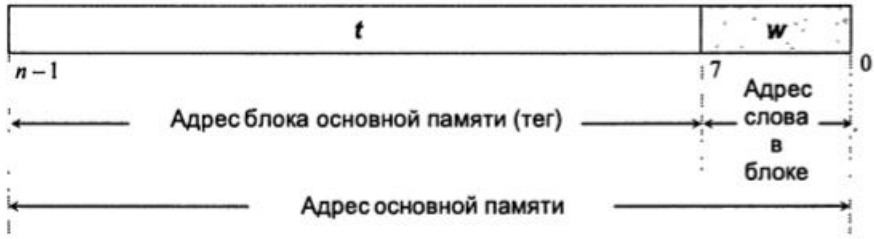


Рис. 3.11. Структура адреса памяти при полностью ассоциативном распределении

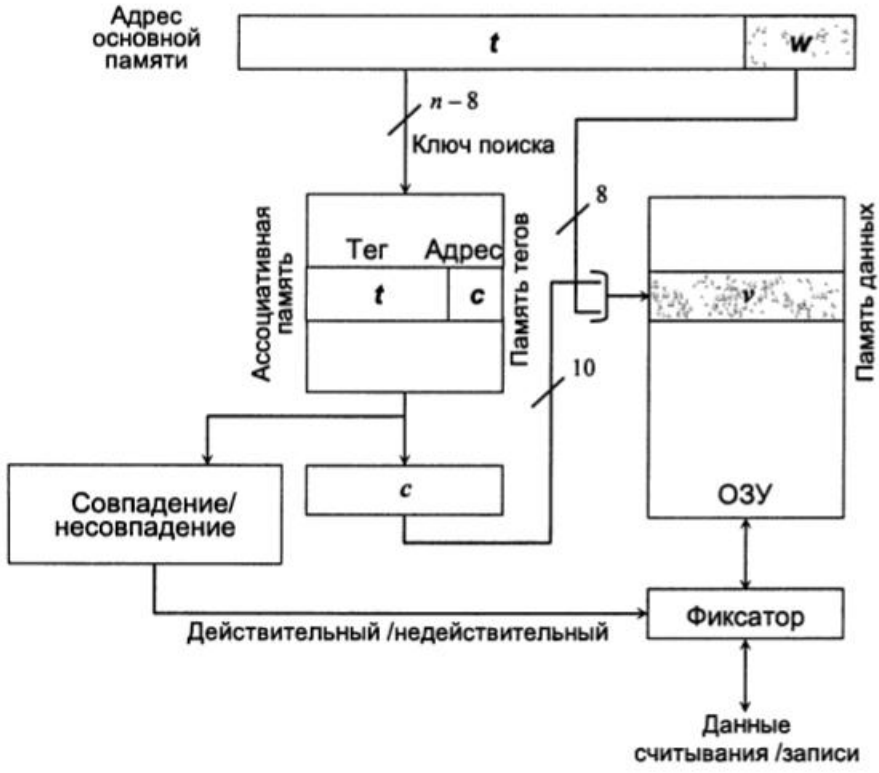


Рис. 3.12. Организация кэш-памяти при полностью ассоциативном распределении

Адрес кэш-строки *c* составляет вместе с адресом слова внутри блока *w* полный адрес искомого слова *v* внутри памяти данных, которое считывается в фик-

сатор. Действительным считанное слово *v* признается только в случае успешного поиска в ассоциативной памяти тегов по ключу *t*.

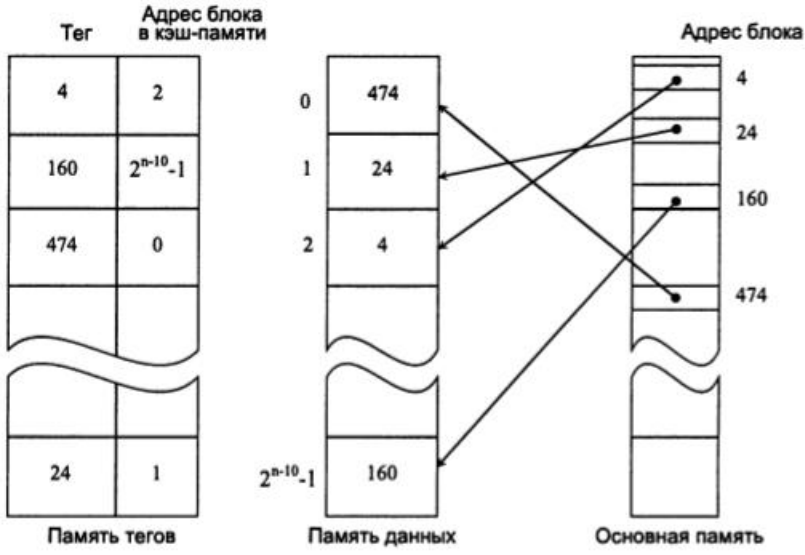


Рис. 3.13. Пример размещения данных при полностью ассоциативном распределении

3.5.1.3. Множественно-ассоциативное распределение

При данном способе несколько соседних блоков (фиксированное число, не менее двух) ОП образуют структуру, называемую **множеством (группой)**. Адрес кэш-строки (блока внутри кэш-памяти) *b* разделяется на **адрес группы** set(*b*) и **адрес блока внутри группы** tag(*b*) (*теговая часть*).

Для помещения в кэш-память копии адреса блока *b*, хранимого в основной памяти, необходимо выбрать группу с адресом set(*b*). При этом не имеет значения, какой из блоков в группе может быть выбран. Для выбора группы используется метод прямого распределения, а для выбора блока в группе— метод полностью ассоциативного распределения.

Если группа состоит из *n* блоков, то такое размещение называется множественно-ассоциативным с *n* каналами (*n*-way set associative). Следует заметить, что диапазон возможных организаций кэш-памяти очень широк: кэш-память с прямым распределением есть просто одноканальная множественно-ассоциативная кэш-память, а полностью ассоциативная кэш-память с *m* блоками может быть названа *m*-канальной множественно-ассоциативной. В современных процессорах, как правило, используется либо кэш-память с прямым отображением, либо двух- (четырех-) канальная множественно-ассоциативная кэш-память.

3.5.2. Стратегии обновления основной памяти и стратегии замещения

Если результат обновления блоков кэш-памяти не возвращать в основную память, то содержимое ОП становится неадекватным вычислительному процессу — нарушается **когерентность** (согласованность копий) памяти. Чтобы избежать этого, предусмотрены методы обновления ОП, которые можно разделить на две большие группы: *методы сквозной записи* и *методы обратной записи*.

При **сквозной записи** (write through, store through) информация записывается одновременно в кэш-строку и в блок основной памяти. При **обратной записи, записи с обратным копированием** (write back, copy back) информация записывается только в кэш-строку. Модифицированная кэш-строка записывается в основную память, только когда она замещается на другой блок ОП. Для сокращения частоты копирования блоков при замещении обычно с каждой кэш-строкой связывается так называемый бит модификации (dirty bit). Этот бит состояния показывает, была ли модифицирована кэш-строка. Если она не модифицировалась, то обратное копирование отменяется, поскольку более низкий уровень содержит ту же самую информацию, что и кэш-память.

Оба подхода к организации записи имеют свои преимущества и недостатки. При записи с обратным копированием операции записи выполняются со скоростью кэш-памяти, и несколько записей в один и тот же блок требуют только одной записи в основную память. Поскольку в этом случае обращения к основной памяти происходят реже, вообще говоря, требуется меньшая полоса пропускания памяти, что очень привлекательно для многопроцессорных систем. Сквозная запись проще для реализации, чем запись с обратным копированием. Преимущество сквозной записи также в том, что основная память имеет наиболее свежую копию данных, что может быть важно в некоторых случаях при совместной работе нескольких процессоров. Подробнее этот вопрос освещается в разделе «Параллельные вычислительные системы».

Когда процессор ожидает завершения записи при выполнении сквозной записи, то говорят, что он приостанавливается для записи (write stall).Общий прием минимизации остановов по записи связан с использованием буфера записи (write buffer), который позволяет процессору продолжить выполнение команд во время обновления содержимого памяти. Следует отметить, что остановы по записи могут возникать и при наличии буфера записи.

При промахе во время записи имеются две дополнительные возможности:

1) разместить записываемый блок в кэш-памяти (write allocate) (называется также *выборкой при записи*, fetch on write). Блок загружается в кэш-память, вслед за чем выполняются действия, аналогичные выполняющимся при выполнении записи с попаданием;

2) не размещать записываемый блок в кэш-памяти (называется также *записью в окружение*, write around). Блок модифицируется в ОП и не загружается в кэш-память.

Обычно в кэш-памяти со сквозной записью используется запись в окружение (поскольку последующая запись в этот блок все равно пойдет в память), а в кэш-памяти с обратной записью используется выборка при записи (в надежде, что последующая запись в этот блок будет перехвачена). Если используется выборка при записи, следует учитывать тот факт, что для размещения в кэш-памяти записываемого блока может не быть свободного места, и, скорее всего, придется применять ту или иную стратегию замещения.

При возникновении промаха чтения контроллер кэш-памяти должен выбрать подлежащий замещению блок. Наиболее простая стратегия замещения используется при прямом распределении. На попадание проверяется только одна кэш-строка, и только эта кэш-строка может быть замещена. При полностью ассоциативной или множественно-ассоциативной организации кэш-памяти необходимо выбрать, какая именно кэш-строка подлежит замещению.

Для замещения кэш-строк применяются три основные стратегии: **случайная**, **FIFO** и **LRU**. В первом случае замещаемая кэш-строка выбирается случайно или псевдослучайно. По методу FIFO (First Input First Output — «первым пришел — первым вышел») среди всех кэш-строк, являющихся объектами замещения, выбирается та, которая была переслана в кэш-память самой первой. При стратегии LRU (Least Recently Used — «не использовался дольше всех») замещению подлежит та кэш-строка, к которой дольше всего не было обращения. Наиболее простой в реализации является стратегия со случайным замещением, наиболее сложной — стратегия FIFO. Наибольшую же эффективность на практике демонстрирует метод LRU.

При реализации этого метода манипуляции с замещаемыми кэш-строками производятся с помощью LRU-стека. При загрузке кэш-строка помещается в этот стек, для замены используется кэш-строка, хранящаяся в наиболее глубокой позиции стека. Именно эта кэш-строка удаляется из стека.

При обращении к определенной кэш-строке она удаляется из стека и заново помещается в него. Таким образом, чем дольше не было доступа к кэш-строке, тем в более глубокой позиции стека она располагается. Реализация LRU-стека, позволяющего с высокой скоростью выполнять манипуляции с кэш-строками, усложняется по мере увеличения числа блоков кэш-памяти и становится дорогой. При множественно-ассоциативном распределении стековым механизмом должна быть оснащена каждая группа кэш-строк, что также ведет к увеличению стоимости аппаратуры.

3.6. АРХИТЕКТУРА ВИРТУАЛЬНОЙ ПАМЯТИ

При работе с физической памятью возникают три проблемы. *Первая* связана, как уже отмечалось, с **ограниченностью** памяти, так как часто при решении многих задач объема памяти не хватает. *Вторая* проблема связана с линейностью адресации памяти. Когда программа по частям загружается в память, то после

завершения ее выполнения и освобождения памяти возникает состояние, называемое **фрагментацией памяти**. Это состояние иллюстрируется рис. 3.14. В этом состоянии в память нельзя загрузить программу, даже имеющую размер меньше суммарного объема свободных областей.

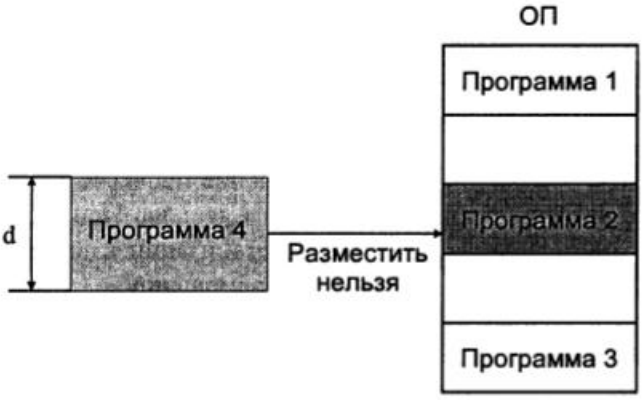


Рис. 3.14. Фрагментация памяти

Наконец, *третья* проблема касается режима мультизадачности, когда память разделяется между несколькими различными программами. При этом необходимо организовать **изоляцию** одной выполняющейся программы от другой. Изоляция должна заключаться в том, что никакая программа не должна иметь возможность читать или изменять участки памяти, принадлежащие другой программе.

Для решения этих проблем используется архитектура памяти, при использовании которой программисту кажется, будто существует достаточно большое пространство памяти, причем необязательно линейное. Данная архитектура получила название **виртуальной памяти (ВП)**. Виртуальная память базируется на идее использования иерархической структуры памяти в качестве непрерывного адресного пространства ОП, доступного для непосредственного размещения программ и данных. Верхним уровнем является физическая ОП (с механизмом кэш-памяти), а нижним — внешняя память (ВЗУ).

Пространство виртуальной памяти, видимое (доступное) программисту, называется **виртуальным пространством**. В отличие от последнего пространство физической памяти называется **реальным пространством**. Адреса виртуального пространства называются **виртуальными адресами**, а адреса физического пространства — **физическими адресами**. Программа составляется с использованием виртуальных адресов, которые потом преобразуются (транслируются) в физические. Этот процесс иллюстрируется рис. 3.15.

Для управления ВП вся память делится на блоки. В ходе вычислительного процесса часть блоков ВП может размещаться в основной, так называемой *первичной*, памяти ЭВМ (эти блоки, как правило, содержат готовые к исполнению программы и/или их данные). Остальные блоки ВП размещаются во внешней па-

мяти, которая называется *вторичной*. В процессе функционирования ЭВМ осуществляется перемещение блоков ВП между ОП и вторичной памятью.

Системы виртуальной памяти можно разделить на два класса: системы с фиксированным размером блоков, называемых **страницами**, и системы с переменным размером блоков, называемых **сегментами**.

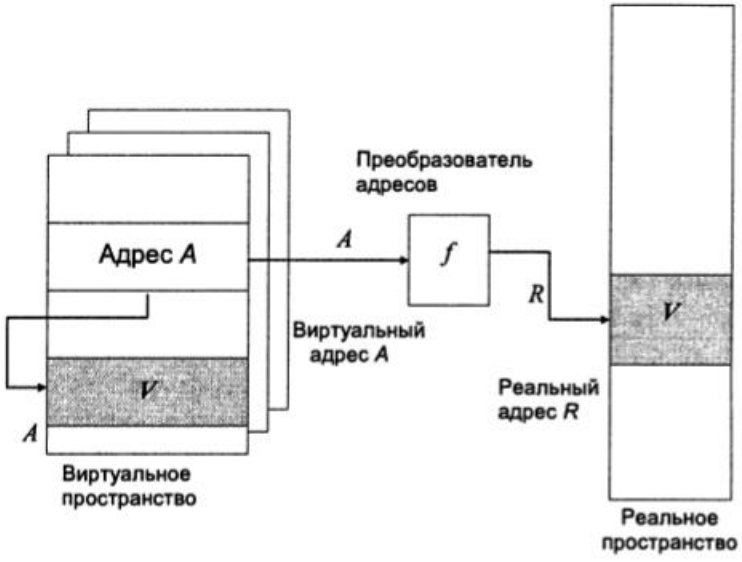


Рис. 3.15. Преобразование виртуального адреса в физический (реальный)

3.6.1. Страничная организация памяти

В системах со страничной организацией основная и внешняя память делятся на блоки или страницы фиксированной длины. Каждому пользователю предоставляется отдельное виртуальное адресное пространство, размер которого ограничен только возможностями адресации, заложенными в системе команд. Размер предоставляемой пользователю виртуальной памяти может превышать размер физической ОП. Каждое слово в виртуальной памяти пользователя определяется виртуальным адресом, состоящим из двух частей: старшие разряды адреса рассматриваются как номер страницы, а младшие — как номер слова (или байта) внутри страницы.

Управление виртуальной памятью осуществляется программами ядра операционной системы, которые следят за распределением страниц и оптимизируют обмены между первичной и вторичной памятью. При страничной организации памяти смежные виртуальные страницы не обязательно должны размещаться на смежных страницах основной физической памяти. Для указания соответствия между виртуальными страницами и страницами основной памяти для каждой

программы операционная система формирует в основной памяти **таблицу страниц**. При этом каждой странице программы, независимо от того, находится ли она в основной памяти или нет, ставится в соответствие некоторый элемент таблицы страниц. Каждый элемент таблицы страниц содержит номер физической страницы основной памяти и специальный индикатор. Единичное состояние этого индикатора свидетельствует о наличии этой страницы в основной памяти. Нулевое состояние индикатора означает отсутствие страницы в оперативной памяти.

Рис. 3.16. Преобразование виртуального адреса в реальный адрес при страничной организации виртуальной памяти

На рис. 3.16 показано преобразование виртуальных адресов в реальные с использованием таблицы страниц.

Символы, используемые на рис. 3.16, имеют следующий смысл:

- бит F (бит существования) показывает, существует или нет i-я страница. Если $F = 1$, то страница существует;
- бит C (бит обновления) показывает, было или нет обновление страницы в ОП. Этот бит нужен для того, чтобы исключить лишнюю запись во вторичную память в случае удаления страницы;
- код RWX определяет форму доступа (считывание/запись или и то и другое) к i-й странице;
- бит M показывает местонахождение страницы. Если $M = 1$, то страница находится в ОП;
- код PFA — номер блока, соответствующий i-й странице.

Если, например, $F = 1$ и $M = 1$, то это означает, что i-я страница хранится в ОП и номер соответствующего блока определяется значением PFA. Если $F = 1$, а $M = 0$, то в PFA хранится адрес страницы из вторичной памяти.

Для увеличения эффективности такого типа схем используется специальная полностью ассоциативная кэш-память, которая также называется **буфером преобразования адресов TLB** (translation-lookside buffer). Заметим, что в целях защиты памяти содержание TLB очищается при переключении с одной программы на другую.

Поиск в таблицах страниц, расположенных в основной памяти, и загрузка TLB могут осуществляться либо программным способом, либо специальными аппаратными средствами. В последнем случае, для того чтобы предотвратить возможность обращения пользовательской программы к таблицам страниц, с которыми она не связана, предусмотрены специальные меры. С этой целью в процессоре выделяется дополнительный регистр защиты, содержащий описатель (дескриптор) таблицы страниц или базово-граничную пару. База определяет адрес начала таблицы страниц в основной памяти, а граница — длину таблицы страниц соответствующей программы. Загрузка этого регистра защиты разрешена только в привилегированном режиме. Для каждой программы операционная система хранит дескриптор таблицы страниц и устанавливает его в регистр защиты процессора перед запуском соответствующей программы.

Отметим некоторые особенности, присущие простым схемам со страничной организацией памяти. Наиболее важной из них является то, что все программы, которые должны непосредственно связываться друг с другом без вмешательства операционной системы, должны использовать общее пространство виртуальных адресов. Это относится и к самой операционной системе, которая, вообще говоря, должна работать в режиме динамического распределения памяти. Поэтому в некоторых системах пространство виртуальных адресов пользователя уменьшается на размер общих процедур, к которым программы пользователей желают иметь доступ. Общим процедурам должен быть отведен определенный объем пространства виртуальных адресов всех пользователей, чтобы они имели постоянное место в таблицах страниц всех пользователей. В этом случае для обеспечения целостности, секретности и взаимной изоляции выполняющихся программ должны быть предусмотрены различные режимы доступа к страницам, которые реализуются с помощью специальных индикаторов доступа в элементах таблиц страниц.

Следствием такого использования является значительный рост таблиц страниц каждого пользователя. Одно из решений проблемы сокращения длины таблиц основано на введении многоуровневой организации таблиц. Частным случаем многоуровневой организации таблиц является сегментация при страничной организации памяти.

3.6.2. Сегментация памяти

Этот подход к организации памяти опирается на возможность разделения программы на отдельные области — сегменты. Каждый сегмент представляет собой отдельную логическую единицу информации, содержащую совокупность данных или кода программы и расположенную в адресном пространстве пользователя. Сегменты создаются пользователями, которые могут обращаться к ним по символическому имени. В каждом сегменте устанавливается своя собственная нумерация слов, начиная с нуля.

Обычно в подобных системах обмен информацией между пользователями строится на базе сегментов. Сегменты являются отдельными логическими единицами информации, которые необходимо защищать, и именно на этом уровне вводятся различные режимы доступа к сегментам. Можно выделить два основных типа сегментов: **программные сегменты и сегменты данных** (сегменты стека являются частным случаем сегментов данных). Поскольку общие программы должны обладать свойством повторной входимости, то из программных сегментов допускается только выборка команд и чтение констант. Запись в программные сегменты может рассматриваться как нарушение защиты и запрещаться системой. Выборка команд из сегментов данных также может считаться нарушением защиты, и любой сегмент данных может быть защищен от обращений по записи или по чтению.

Для реализации сегментации существуют несколько схем, которые отличаются друг от друга деталями реализации, но основаны на одних и тех же принципах. При сегментации памяти каждое слово в адресном пространстве пользователя определяется виртуальным адресом, состоящим из двух частей: старшие разряды адреса рассматриваются как номер сегмента, а младшие — как номер слова внутри сегмента. Наряду с сегментацией может также использоваться страничная организация памяти. В этом случае виртуальный адрес слова состоит из трех частей: старшие разряды адреса определяют номер сегмента, средние — номер страницы внутри сегмента, а младшие — номер слова внутри страницы.

Как и в случае страничной организации, необходимо обеспечить преобразование виртуального адреса в физический адрес основной памяти. С этой целью для каждого пользователя операционная система должна сформировать таблицу сегментов. Каждый элемент таблицы сегментов содержит описатель (дескриптор) сегмента (поля базы, границы и индикаторов режима доступа). При отсутствии страничной организации поле базы определяет адрес начала сегмента в основной памяти, а граница — длину сегмента. При наличии страничной организации поле базы определяет адрес начала таблицы страниц данного сегмента, а граница — число страниц в сегменте. Поле индикаторов режима доступа представляет собой некоторую комбинацию признаков блокировки чтения, записи и выполнения.

Таблицы сегментов различных пользователей операционная система хранит в основной памяти. Для определения расположения таблицы сегментов выпол-

няющейся программы используется специальный регистр защиты, который загружается операционной системой перед началом ее выполнения. Этот регистр содержит дескриптор таблицы сегментов (базу и границу), причем база содержит адрес начала таблицы сегментов выполняющейся программы, а граница — длину этой таблицы сегментов. Разряды номера сегмента виртуального адреса используются в качестве индекса для поиска в таблице сегментов. Таким образом, наличие базово-граничных пар в дескрипторе таблицы сегментов и элементах таблицы сегментов предотвращает возможность обращения программы пользователя к таблицам сегментов и страниц, с которыми она не связана. Наличие в элементах таблицы сегментов индикаторов режима доступа позволяет осуществить необходимый режим доступа к сегменту со стороны данной программы. Для повышения эффективности схемы используется ассоциативная кэш-память.

Пример преобразования адресов с помощью сегментной таблицы приведен на рис. 3.17.

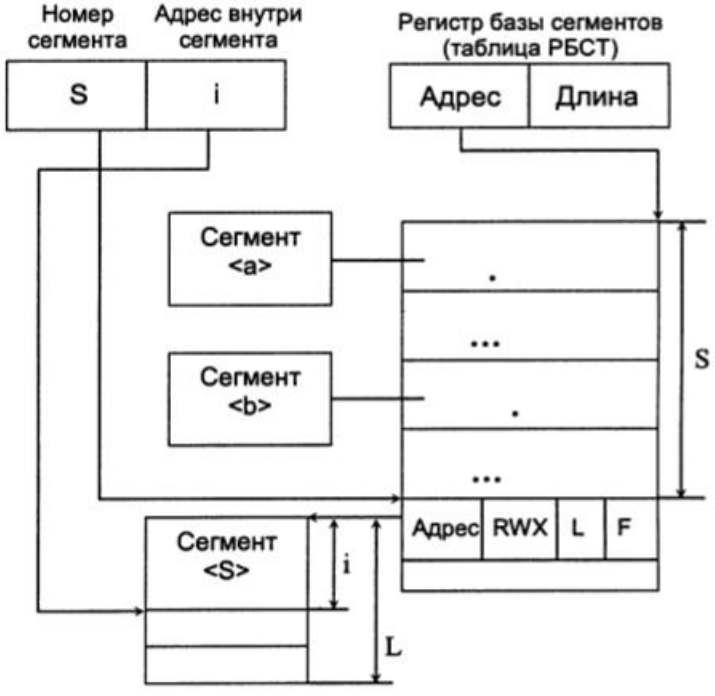


Рис. 3.17. Преобразование адресов с помощью сегментной таблицы

Каждый сегмент (*S*) виртуального пространства имеет в таблице свой адрес (*S*), по которому хранится вся информация, относящаяся к данному сегменту. Эта информация включает:

- тип доступа (поле RWX) к сегменту;

• местоположение сегмента (поле F, если $F = 1$, то сегмент S находится в ОП, если $F = 0$, то во вторичной памяти;

• начальный адрес сегмента и размер сегмента (поле L).

Выборка и замена сегментов выполняется аналогично выборке и замене страниц. Реальный адрес вычисляется путем добавления к начальному адресу, полученному из таблицы сегментов, адреса внутри сегмента.

Отметим, что в рассмотренной схеме сегментации таблица сегментов с индикаторами доступа предоставляет всем программам, являющимся частями некоторой задачи, одинаковые возможности доступа, т. е. она определяет единственную область (домен) защиты. Однако для создания защищенных подсистем в рамках одной задачи, чтобы изменять возможности доступа, когда точка выполнения переходит через различные программы, управляющие ее решением, необходимо связать с каждой задачей множество доменов защиты. Реализация защищенных подсистем требует разработки некоторых специальных аппаратных средств.

Вопросы для самоконтроля

1. Дайте определения основным характеристикам памяти.
2. Какие существуют методы доступа к памяти и чем они отличаются друг от друга?
3. Какую проблему и за счет чего решает иерархическая организация памяти?
4. В чем состоит принцип локальности по обращению?
5. На каких принципах строятся оперативные запоминающие устройства?
6. За счет чего может быть повышено быстродействие ядра микросхем ОЗУ?
7. Как можно оптимизировать доступ к ОЗУ?
8. Каковы назначение и структура блочной организации ОП?
9. Для чего и как применяется расслоение памяти?
10. Охарактеризуйте пакетный и конвейерный методы доступа к памяти.
11. Как устроена ассоциативная память?
12. Как работает кэш-память с прямым распределением?
13. Как работает кэш-память с полностью и множественно-ассоциативным распределением?
14. Охарактеризуйте стратегии обновления ОП и замещения блоков в кэш-памяти.
15. Для чего нужна виртуальная память?
16. В чем заключается страничная организация виртуальной памяти?
17. Как организуется сегментация виртуальной памяти?

ГЛАВА 4. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА

4.1. ОБЩАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ ВВОДА-ВЫВОДА

Довольно часто при оценке производительности ЭВМ или ВС используются только оценки производительности процессора, а влиянием системы ввода-вывода на общую производительность системы пренебрегают. Такое отношение не совсем правильно, поскольку подсистема ввода-вывода во многом определяет такую важнейшую характеристику производительности, как **время ответа** (время между моментом ввода пользователем задания и получением им результата), которое учитывает все накладные расходы, связанные с выполнением задания в системе, включая ввод-вывод.

Как уже отмечалось, в ЭВМ различных классов часто используются одни и те же типы микропроцессоров. Отличия ЭВМ одного класса от ЭВМ другого класса (например, настольной ЭВМ от сервера) во многом определяются различной организацией системы ввода-вывода.

Основной функцией системы ввода-вывода является организация информационного обмена между центральной частью ЭВМ (процессор, память) и периферийными (внешними) устройствами — дисковыми и иными накопителями, принтерами, средствами ввода информации и т. п. Заметим, что имеет место огромное разнообразие как организации центральной части ЭВМ, так и самих внешних устройств (ВУ). Сопряжение этих различных устройств осуществляется системой ввода-вывода, которая для этой цели реализуется в виде совокупности **модулей ввода-вывода (МВВ)**.

4.1.1. Структура и функции модуля ввода-вывода

Каждый МВВ, с одной стороны, тем или иным образом подключается к процессору и памяти (так называемый **«большой» интерфейс**), а с другой стороны — к одному или нескольким ВУ (так называемый **«малый» интерфейс**). Рассмотрим структуру и функции МВВ подробнее. К основным функциям МВВ относятся:

- локализация данных;
- управление и синхронизация;
- обмен информацией;
- буферизация данных;
- обнаружение и исправление ошибок.

Локализация данных подразумевает возможность обращения к одному из ВУ и адресации данных на нем. Обращение к ВУ осуществляется с помощью **адреса**, состоящего, как правило, из адреса МВВ и адреса ВУ, подключенного к данному МВВ. Для простейших ВУ (клавиатура, принтер и т. п.) адрес ВУ однозначно определяет и расположение данных в этом устройстве. Для сложных ВУ информация о местонахождении данных требует детализации. Например, для дис-

кового накопителя необходимо указать номер дорожки, номер сектора и смещение информации внутри сектора.

Функция **управления и синхронизации** заключается в том, что МВВ должен координировать перемещение данных между центральной частью ЭВМ и внешними устройствами. Процессор может взаимодействовать одновременно с несколькими ВУ с различным быстродействием, при этом процессы ввода-вывода и работа процессора протекают не синхронно. Очередная порция информации может быть выдана на устройство вывода лишь тогда, когда это устройство готово к приему. Аналогично ввод из устройства ввода может быть осуществлен только в случае доступности информации на ВУ. На МВВ возлагается обязанность информировать процессор о собственной готовности и о готовности подключенных к модулю ВУ.

Общий алгоритм **обмена информацией** заключается в следующем.

1. Выбор требуемого ВУ.
2. Определение состояния ВУ.
3. Логическое подключение нужного ВУ к процессору.
4. Распознавание сигнала о готовности ВУ к передаче очередной порции информации.
5. Прием (передача) информации.
6. Циклическое повторение пп. 4 и 5 до завершения передачи полного объема информации.
7. Логическое отсоединение ВУ от процессора.

Буферизация информации призвана компенсировать различие в скорости работы ВУ и центральной части ЭВМ. Каким бы скоростным ни было ВУ, скорость информационного обмена с ним всегда будет ниже, чем скорость обмена «процессор-память». При выводе информации данные с высокой скоростью пересылаются из основной памяти в МВВ, где буферизуются, а затем направляются во ВУ с медленной скоростью. При вводе из ВУ данные сначала накапливаются в буфере МВВ, а затем быстро пересылаются в ОП.

Обнаружение ошибок является одной из важнейших функций МВВ. Внешние устройства в большей степени, чем центральная часть ЭВМ, подвержены воздействию внешней среды. Кроме этого, большинство ВУ содержат подверженные износу механические части, надежность которых на несколько порядков ниже, чем у электронных компонент.

Обобщенная структура МВВ представлена на рис. 4.1. Данные, передаваемые в МВВ и из него, буферизуются в буфере данных. Разрядность ячеек буфера, как правило, совпадает с шириной шины данных «большого» интерфейса. Поскольку значительное количество ВУ поддерживает побайтный обмен, МВВ содержит узел упаковки/распаковки данных. При вводе информации данный узел обеспечивает последовательное заполнение буфера (упаковку), а при выводе — последовательную побайтную передачу данных в ВУ (распаковку). При обмене данными через

«большой» интерфейс занимается вся ширина шины данных, что существенно повышает эффективность системы ввода-вывода.

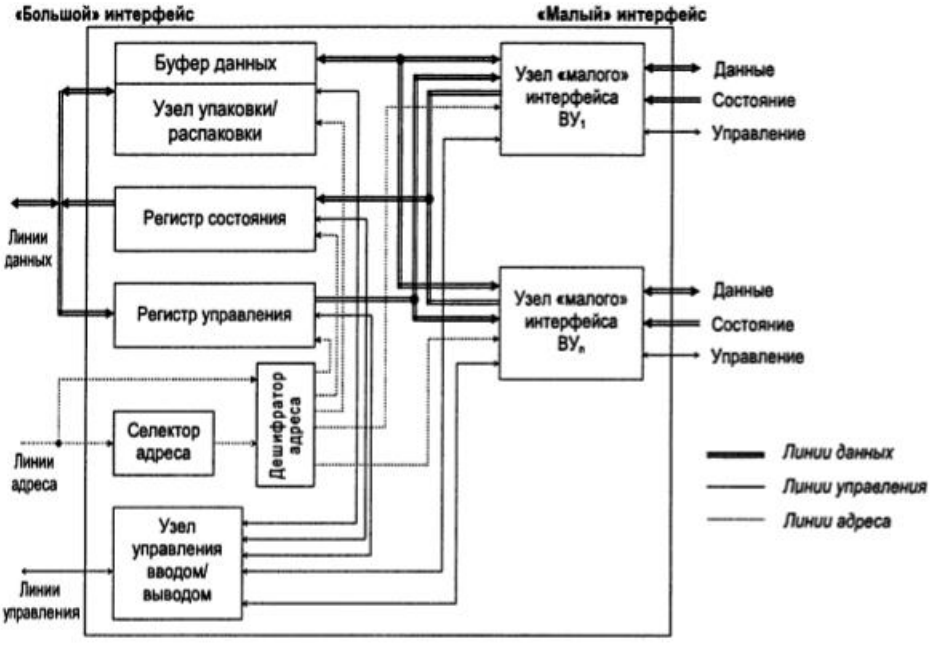


Рис. 4.1. Структура модуля ввода-вывода

В регистре управления (РУ) фиксируются поступившие из процессора команды управления МВВ или подключенными к нему ВУ. Запись единицы или нуля в отдельные разряды РУ может означать различные управляющие воздействия — сброс ВУ и МВВ, начало чтения, начало записи и т. п. Регистр состояния (РС) определяет состояние МВВ и ВУ. Отдельные разряды РС могут характеризовать, например, готовность ВУ к принятию очередной порции информации, занятость ВУ и т. п. Сложные МВВ могут содержать несколько РУ и РС, например по одному для каждого ВУ.

Как и обращение к памяти, операции ввода-вывода предполагают наличие некоторой системы адресации, позволяющей выбрать один из модулей ввода-вывода, а также одно из подключенных к нему ВУ. Совокупность всех возможных адресов модулей ввода-вывода и внешних устройств образует **адресное пространство системы ввода-вывода**, которое может быть либо *совмещенным* с адресным пространством основной памяти, либо *выделенным*.

При **совмещении адресного пространства** для адресации МВВ и ВУ используется определенная часть адресов основной памяти. Поскольку процедура ввода-вывода фактически сводится к записи в одни регистры МВВ и считыванию из других, это позволяет рассматривать регистры МВВ как ячейки основной памяти.

Обращение к регистрам МВВ осуществляется аналогично обращению к ОП, при этом в системе команд ЭВМ могут вообще отсутствовать команды ввода-вывода. Примерами использования совмещенного адресного пространства являются ЭВМ MIPS и SPARC.

В случае **выделенного адресного пространства** для обращения к МВВ и ВУ применяются специальные команды и отдельная система адресов. Выделенное адресное пространство используется в подавляющем большинстве современных ЭВМ.

Вне зависимости от характера адресного пространства каждому МВВ выделяется уникальных набор адресов (диапазон). Поступивший в МВВ адрес с помощью селектора адреса проверяется на вхождение в диапазон, выделенный данному МВВ. В случае подтверждения дешифратор адреса производит его декодирование, разрешая работу с соответствующими регистрами МВВ или с ВУ.

На узел управления вводом-выводом возлагаются задачи обеспечения взаимодействия с центральной частью ЭВМ и координации работы всех составляющих МВВ. Из центральной части ЭВМ по линиям управления в МВВ поступают сигналы, служащие для синхронизации операций чтения и записи. В обратном направлении передаются сигналы, информирующие о состоянии МВВ, например сигналы прерываний.

Со стороны «малого» интерфейса МВВ обеспечивает подключение ВУ и взаимодействие с ними. Каждое из периферийных устройств обслуживается своим узлом «малого» интерфейса, который реализует принятый для данного ВУ стандартный протокол взаимодействия.

При управлении широким спектром ВУ модуль должен по возможности освобождать ЦП от знания деталей конкретных ВУ, так чтобы процессор мог управлять любым устройством с помощью простых команд ввода-вывода. МВВ при этом берет на себя задачи синхронизации, согласования форматов данных и т.п. Модуль ввода-вывода, который берет на себя детальное управление ВУ и общается с процессором только с помощью команд высокого уровня, называют **каналом** или **процессором ввода-вывода**. Более примитивный МВВ, требующий от ЦП детального управления ВУ, называется **контроллером ввода-вывода,** или **контроллером ВУ**.

Как правило, контроллеры ввода-вывода подсоединяются к центральной части ЭВМ посредством **общей шины**. Такая организация системы ввода-вывода получила название **шинной**. Альтернативой является использование каналов (процессоров) ввода-вывода, такая архитектура называется **канальной организацией**.

4.1.2. Методы управления вводом-выводом

Существуют три основных метода управления вводом-выводом.

1. Ввод-вывод с опросом.

2. Ввод-вывод по прерываниям.
3. Прямой доступ к памяти.

Наиболее простым методом управления является **ввод-вывод с опросом**, который происходит под полным контролем центрального процессора и реализуется специальной программной процедурой ввода-вывода.

Процедура заключается в последовательной выдаче процессором команд ввода-вывода, в которых указывается адрес конкретного МВВ и конкретного ВУ. Различают четыре типа команд ввода-вывода: *управление*, *проверка*, *чтение* и *запись*.

Команды управления используются для активизации ВУ и указания требуемой операции. Например, для дискового накопителя может быть выдана команда перемещения на одну дорожку. Для каждого типа ВУ характерны типичные для него команды управления.

Команды проверки применяются для определения состояний МВВ и ВУ. С помощью таких команд проверяется, включено ли ВУ, готово ли оно к работе, завершена ли последняя операция ввода-вывода, и не возникли ли в ходе ее выполнения какие-либо ошибки. Действие команды часто сводится к чтению соответствующих разрядов регистра состояния МВВ.

По **команде чтения** МВВ получает элемент данных из ВУ и заносит его в буфер данных. Процессор получает данные, запросив модуль поместить содержимое буфера на шину данных «большого» интерфейса.

Команда записи заставляет МВВ принять элемент данных с шины данных «большого» интерфейса и занести его в буфер данных с последующей пересылкой в ВУ.

Рассмотрим процедуру ввода-вывода на примере записи блока данных в ВУ.

1. Процессор выдает соответствующему МВВ команду начала вывода данных.
2. Процессор читает регистр состояния МВВ и определяет, готово ли соответствующее ВУ к приему данных. Данный пункт повторяется циклически до тех пор, пока ВУ не будет готово к операции записи. В случае ошибки ВУ процедура завершается.
3. Процессор выполняет чтение очередного слова из основной памяти. При этом проверяется счетчик записанных слов, полностью ли выведен в ВУ заданный блок информации. Если достигнут конец блока, процедура завершается.
4. Процессор записывает в буфер данных МВВ очередное слово блока.
5. Осуществляется переход к п. 1.

Преимуществом ввода-вывода с опросом является *простота системы ввода-вывода*, поскольку основные функции управления возлагаются на процессор. Кроме этого, при одновременной работе с несколькими ВУ, программными средствами легко устанавливаются и меняются приоритеты обслуживания. Для

изменения порядка приоритетов достаточно соответствующим образом изменить порядок опроса ВУ.

Главным *недостатком* рассмотренного метода, во многом перечеркивающим преимущества, является *крайне неэффективное использование процессора*. Помимо того, что процессор вынужден простаивать из-за ожидания готовности ВУ, передача даже одного слова требует выполнения нескольких команд (чтение и анализ битов регистра состояния МВВ, запись в регистр управления МВВ, чтение или запись данных в буфер данных).

Альтернативным решением может быть **ввод-вывод по прерываниям**. При этом подходе процессор выдает команду ввода-вывода в МВВ, а затем продолжает вычисления. Когда ВУ становится готово к обмену данными, оно через МВВ извещает об этом процессор, выставляя **запрос на прерывание**. Обрабатывая запрос, процессор прерывает вычисления, осуществляет передачу очередного элемента данных, после чего возобновляет прерванные вычисления.

Рассмотрим процедуру ввода-вывода по прерываниям на примере чтения информации из ВУ.

1. Процессор инициализирует обмен путем выдачи соответствующей команды в МВВ, после чего занимается другим вычислениями.
2. МВВ, получив команду, приступает к вводу элемента данных из ВУ. Когда буфер данных полностью заполняется, МВВ выставляет запрос на прерывание к процессору.
3. Процессор обрабатывает запрос на прерывание. Для этого сохраняется контекст текущей программы и вызывается соответствующая процедура-обработчик прерывания, которая выполняет следующие действия.
   3.1. Читается содержимое регистра состояния МВВ. Состояние ВУ проверяется на предмет наличия ошибок передачи, в случае обнаружения которых процедура ввода-вывода завершается.
   3.2. Производится чтение элемента данных из буфера данных МВВ.
   3.3. Производится запись прочитанного элемента в основную память.
   3.4. Производится проверка на завершение чтения заданного блока данных (все ли данные считаны). Если блок считан полностью, процедура ввода-вывода завершается.
   3.5. В МВВ передается команда ввода очередной порции данных.
4. Процессор восстанавливает контекст прерванной программы, после чего циклически повторяются пп. 2 и 3.

Порядок определения источника прерывания и передачи управления программе-обработчику подробно рассмотрен в разделе 2.6 «Система прерываний программ ЭВМ».

Метод ввода-вывода по прерываниям заметно эффективнее ввода-вывода с опросом, поскольку процессор не ожидает готовности ВУ. Тем не менее каждый

элемент вводимых или выводимых данных проходит через процессор, при этом обработка прерывания занимает достаточно много времени.

Поскольку в обоих рассмотренных методах (с опросом и по прерываниям) на процессор полностью возложены все функции управления и все данные проходят через процессор, эти методы получили название **процессорного ввода-вывода (PIO — Processor Input-Output)**.

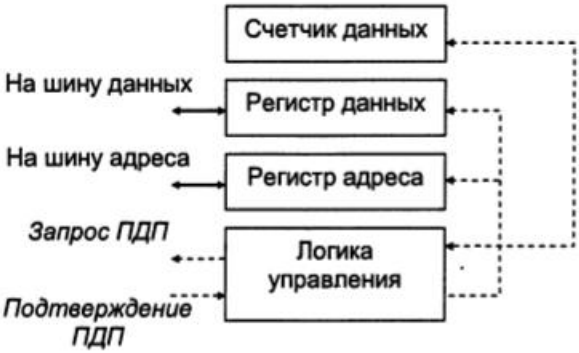


Рис. 4.2. Схема контроллера ПДП

Альтернативой PIO является метод ввода-вывода с **прямым доступом к памяти (ПДП)**. ПДП предполагает наличие в системе ввода-вывода специального устройства — **контроллера прямого доступа к памяти (КПДП**, рис. 4.2). КПДП способен брать на себя функции процессора по управлению вводом-выводом и обеспечивать пересылку информации напрямую между основной памятью и МВВ. Во все время информационного обмена между МВВ и ОП процессор может выполнять другие вычисления.

Процедура ввода-вывода с использованием ПДП выглядит следующим образом (на примере операции чтения из ВУ).

1. Процессор инициализирует КПДП, сообщая ему следующие параметры:
   - вид запроса (чтение или запись, в нашем случае запись в ОП);
   - адрес МВВ и ВУ;
   - адрес начальной ячейки блока основной памяти, куда будет заноситься информация, помещается в регистр адреса КПДП;
   - количество слов, подлежащих записи в ОП, помещается в счетчик данных КПДП.
2. После инициализации процесс пересылки информации может быть начат в любой момент. ВУ извещает контроллер ПДП о своей готовности к обмену.
3. КПДП посылает процессору сигнал «запрос ПДП». Процессор освобождает шину основной памяти и делегирует право на ее использование контроллеру ПДП посылкой сигнала «подтверждение ПДП».

4. КПДП считывает из ВУ очередной элемент данных, заполняя регистр данных.

5. Когда регистр данных заполнен, КПДП выставляет на шину адреса содержимое регистра адреса, а на шину данных — регистра данных, выполняя запись очередного слова в ОП.

6. После пересылки каждого слова логика управления прибавляет единицу к содержимому регистра адреса (формирует адрес следующей ячейки ОП) и уменьшает на единицу содержимое счетчика данных (ведет подсчет переданных слов).

7. КПДП повторяет пп. 4–6 до обнуления счетчика данных, после чего снимает сигнал «запрос ПДП», освобождая шину памяти. В ответ процессор снимает сигнал «подтверждение ПДП».

Как видно из приведенного примера, процессор вовлекается в процесс ввода-вывода только в начале и в конце передачи данных. Эффективность ПДП зависит от того, как реализовано распределение шины памяти между процессором и КПДП в процессе пересылки блока данных. Здесь может применяться один из трех режимов:

1) блочная пересылка;
2) пропуск цикла;
3) прозрачный режим.

При **блочной пересылке** КПДП полностью захватывает шину памяти с момента начала обмена до его завершения. Данный подход наиболее прост в реализации. Однако на весь период передачи блока данных процессор не имеет доступа к основной памяти, что также может вызвать простои и снижение производительности.

В режиме **пропуска цикла** КПДП после передачи каждого слова на один цикл шины освобождает шину памяти, предоставляя ее на это время процессору. Поскольку КПДП все равно должен ожидать готовности ВУ, процессор может воспользоваться предоставляемыми циклами шины без существенного снижения скорости ввода-вывода.

В **прозрачном режиме** КПДП имеет доступ к шине памяти только в тех циклах, когда в ней не нуждается процессор. При этом подходе обеспечивается наиболее эффективная работа процессора, но может существенно снизиться скорость ввода-вывода. Заметим, что в прозрачном режиме шина памяти может быть предоставлена КПДП в середине цикла команды, т. е. между такими микрооперациями, как выборка команды, декодирование команды, выборка операндов, исполнение команды, запись результата.

4.2. КАНАЛЬНАЯ ОРГАНИЗАЦИЯ ПОДСИСТЕМ ВВОДА-ВЫВОДА

Эта форма организации ввода-вывода широко используется в мэйнфреймах, начиная с больших универсальных ЭВМ 3-го поколения. Данный способ обеспечивает мультипрограммный режим работы ВС, когда одновременно с обработкой данных выполняются процедуры ввода и вывода данных.

Логически такая подсистема ввода-вывода представляет собой совокупность подканалов обмена между внешними устройствами и ОП; каждому внешнему устройству выделяется логический **подканал**.

Реальные подканалы могут быть двух видов: **неразделенные** и **разделенные**. *Неразделенный подканал* представляет собой средство выполнения процедуры обмена с определенным ВУ. *Разделенный подканал* — средство, которое может использоваться попеременно различными логическими подканалами для организации обмена с многими ВУ. Разделенный подканал может иметь собственное оборудование и отдельное сопряжение с ВУ.

Управление передачей данных между ОП и ВУ осуществляет канал ввода-вывода (или просто канал).

Канал может работать в двух режимах: **монопольном** и **мультиплексном**. При монопольном режиме в процессе ввода-вывода обслуживается только одно устройство, работа остальных ВУ блокируется. В мультиплексном режиме каждое из присоединенных ВУ попеременно взаимодействует с каналом в интервалы времени, устанавливаемые автоматически.

Мультиплексный (иногда называют **байт-мультиплексный**) канал имеет определенное число **неразделенных подканалов**, т. е. способен одновременно обеспечить выполнение операций ввода-вывода в каждом подканале, при этом общая нагрузка не должна превышать его пропускную способность.

Для обмена с быстрыми ВЗУ в монопольном режиме используют **селекторный канал** (иногда называют **блоковый**, или **блок-мультиплексный канал**).

Функционально подсистема ввода-вывода включает в свой состав **процессоры ввода-вывода (канальные процессоры)**, **устройства управления внешними устройствами (контроллеры)** и собственно **ВУ**.

Канальный процессор, являющийся по сути специализированной ЭВМ, инициализирует работу канала с помощью команд ввода-вывода, в которых указывается вид операции, адрес канала и внешнего устройства, с которым будет происходить обмен.

Обычно таких команд в системе команд ЭВМ несколько. Например, для мэйнфреймов фирмы IBM (S/370, S/390 и др.) имеется не менее 8 команд ввода-вывода:

1) SIO — начать ввод-вывод;
2) SIOF — начать ввод-вывод с быстрым отключением;
3) TCH — проверить (опросить) канал;
4) TIO — проверить (опросить) ввод-вывод;
5) CLRIO — освободить ввод-вывод;
6) HIO — остановить ввод-вывод;
7) HDV — остановить устройство;
8) STIDC — записать идентификатор канала.

На рис. 4.3 приведен формат команд ввода-вывода для указанных выше семейств ЭВМ.

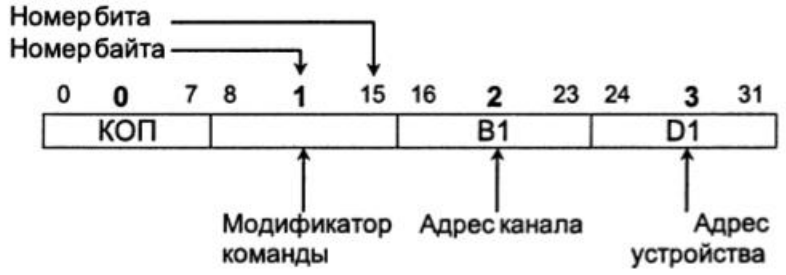


Рис. 4.3. Формат команды ввода-вывода

Как следует из рис. 4.3, в поле адресов (байты 2 и 3) можно указывать до 256 каналов и до 256 устройств.

Ниже приведены примеры записи команд ввода-вывода.

```
SIO WWOD ; адрес канала и устройства задан символическим
именем.
SIO X'00C' ; адресован канал с номером 0, а адрес УВВ
равен 0C.
```

Для того чтобы задать устройству ВВ необходимую команду, указать откуда брать данные для ввода-вывода и сколько данных необходимо вводить (выводить), надо указать в программе на языке ассемблера команду CCW, имеющую формат:

имя	CCW	Четыре операнда, разделенные запятыми

Операнд 1 — абсолютное выражение, определяющее код команды.

Операнд 2 — абсолютное или перемещаемое выражение, определяющее адрес данных (в 16-ричной системе счисления).

Операнд 3 — абсолютное выражение, определяющее признаки (флажки).

Операнд 4 — абсолютное выражение, определяющее количество (в 16-ричной системе счисления) информации в байтах, участвующих в операции.

Например, команда

```
READ CCW X'02', X'800', X'00', X'50'
```

осуществляет считывание (код 02), 80 байтов (50_{16}), в память с адреса 800_{16}.

На рис. 4.4 приведен вариант организации подсистемы ввода-вывода с 6-ю каналами (1 мультиплексный и 5 селекторных).

Для управления работой внешних устройств в ОС входит супервизор ввода-вывода и набор стандартных методов доступа, обеспечивающих обработку запросов пользователя, подготовку программ канала и обработку наборов данных. Управление операциями ввода-вывода осуществляется на физическом уровне.

В большинстве ЭВМ интерфейсы между ЦП и каналами идентичны с точки зрения состава передаваемой информации и логики ее обработки. В канальных

подсистемах ввода-вывода используют для подключения ВУ как радиальные (магистральные), так и шинные интерфейсы. Возможно использование и комбинированных интерфейсов.

Главным достоинством канальной архитектуры является ее масштабируемость, позволяющая практически неограниченно наращивать производительность подсистемы ввода-вывода. К недостаткам следует отнести высокую стоимость, сложность и громоздкость системы.

Рис. 4.4. Структурная схема канальной подсистемы ввода-вывода

4.3. ШИННАЯ ОРГАНИЗАЦИЯ ПОДСИСТЕМЫ ВВОДА-ВЫВОДА

4.3.1. Общая организация шин

Особенность данной организации ввода-вывода заключается в использовании *общей информационной магистрали* для передачи данных между процессором, памятью и контроллерами (устройствами) ввода-вывода (УВВ), подключенными к шине. Заметим, что термин «шина» применительно к системе ввода-вывода используется для обозначения информационных магистралей как со стороны «большого», так и со стороны «малого» интерфейса. Если речь идет о шинной архитектуре системы ввода-вывода, очевидно, имеется в виду «большой» интерфейс, организованный в виде общей шины (рис. 4.5).

Как правило, шины состоят только из пассивных элементов, и все управление передачами выполняется передающим и принимающим устройствами. Так как между разными устройствами возникает состязание за пользование общим ресурсом, то необходимо управление предоставлением шины передающему устройству.

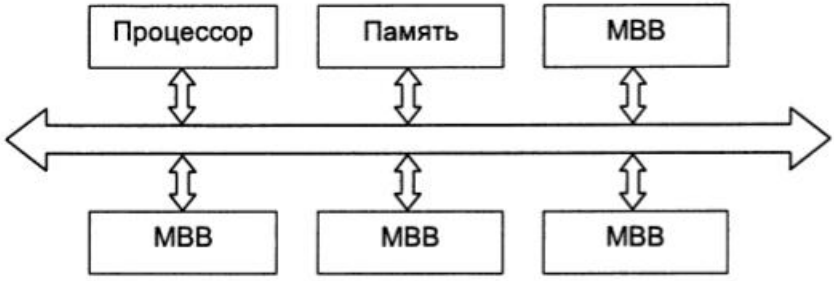


Рис. 4.5. Шинная организация подсистемы ввода-вывода

С целью снижения стоимости некоторые ЭВМ имеют единственную шину для памяти и устройств ввода-вывода. Такая шина называется **системной**. Примерами системных шин являются шины стандартов **ISA**, **EISA** или **MCA** персональных компьютеров недалекого прошлого. Необходимость сохранения баланса производительности по мере роста быстродействия процессоров привела к двухуровневой организации шин в персональных компьютерах на основе *локальной шины*. **Локальной** называется шина, электрически выходящая непосредственно на контакты процессора. Она обычно объединяет процессор, память, схемы буферизации для системной шины и ее контроллер, а также некоторые вспомогательные схемы. Типичными примерами локальных шин являются **VL-Bus** и **PCI**.

Устройство, желающее передать данные, должно сначала получить доступ к шине, затем установить связь с адресатом и определить его способность к взаимодействию. После этого передающее устройство извещает принимающее о действиях, которые должно совершить принимающее устройство в ходе взаимодействия. Принимающее устройство распознает свой адрес на шине и отвечает на управляющие сигналы от передающего. Только после этого происходит собственно передача данных.

Передача элемента данных по шине от одного МВВ к другому осуществляется в виде неделимой последовательности операций, называемой **транзакцией**. Шинная транзакция включает в себя два этапа: *посылку адреса* и *прием* (*или посылку*) *данных*. Во время интервала между посылкой адреса и получением данных процессор или МВВ, инициировавший обмен, могут *простаивать*.

Подключенное к общей шине устройство, которое может инициировать транзакции чтения или записи, называется **главным устройством шины**. ЦП, например, всегда является главным устройством шины. Шина может иметь несколько главных устройств, например, если имеется несколько ЦП или когда МВВ могут инициировать транзакции на шине. Если имеется несколько таких устройств, то требуется **схема арбитража**, чтобы решить, кто следующий «захватит» шину.

По способу коммутации различают два типа шин: *шины с коммутацией цепей* (*circuit-switched bus*) и *шины с коммутацией пакетов* (*packet-switched bus*), получившие свои названия по аналогии со способами коммутации в сетях передачи данных. Шина с **коммутацией пакетов** обеспечивает большую пропускную спо-

собность по сравнению с шиной с коммутацией цепей за счет разделения транзакции на две логические части: *запроса шины* и *ответа*. Каждая часть транзакции помечается (тегируется) так, чтобы при их разделении можно было бы определить, к какой транзакции относится та или иная часть. Произвольное количество других пакетов или транзакций могут использовать шину между запросом и ответом, поэтому разделенная на две части транзакция называется **расщепленной**.

Шина **с коммутацией цепей** не делает расщепления транзакций, любая транзакция на ней есть неделимая операция. Главное устройство запрашивает шину, после арбитража помещает на нее адрес и блокирует шину до окончания обслуживания запроса. При этом часть времени обслуживания тратится на второстепенные операции на шине, например на задержку выборки из памяти.

Расщепленные транзакции делают шину доступной для других главных устройств в то время, пока идет информационный обмен. Шины с расщеплением транзакций имеют более высокую пропускную способность, но при этом, как правило, и большую задержку, чем шины, которые захватываются на все время выполнения транзакции.

По типу синхронизации шины делятся на *синхронные* и *асинхронные*. **Синхронная** шина включает сигналы синхронизации, которые передаются по линиям управления шины, и фиксированный протокол, определяющий расположение сигналов адреса и данных относительно сигналов синхронизации. Простота подобной организации позволяет получать производительные решения с невысокой стоимостью. Однако в синхронных шинах возникает проблема перекоса синхросигналов, ограничивающая физическую длину шины. Обычно синхронные шины используются для организации обмена между ЦП и памятью.

Асинхронная шина обычно использует старт-стопный режим передачи и протокол «**рукопожатия**» (**handshaking**) между источником и приемником данных на шине. Эта схема менее быстродействующая, но позволяет использовать длинные шины. Выбор типа шины (синхронной или асинхронной) определяет не только пропускную способность, но также непосредственно влияет на емкость системы ввода-вывода в терминах физического расстояния и количества УВВ, которые могут быть подсоединены к шине.

Шинная организация ввода-вывода имеет два основных *преимущества*: **низкую стоимость** и **универсальность**. К общей шине легко могут быть подсоединены новые УВВ, и одни и те же периферийные устройства можно применять в ЭВМ разной архитектуры, но использующих однотипную шину. Стоимость такой организации получается достаточно низкой, поскольку для реализации множества путей передачи информации используется единственный набор линий шины, разделяемый множеством устройств.

Главным *недостатком* организации с общей шиной является то, что шина создает **узкое место**, ограничивая как максимальную пропускную способность системы ввода-вывода, так и ее масштабируемость. Количество внешних устройств, которые одновременно могут быть подключены к общей шине весьма ограниченно. В

серверах, где ввод-вывод осуществляется очень часто, одним из главных вопросов разработки является создание системы нескольких шин, способной удовлетворить все запросы.

Отметим одну из существенных особенностей шинной организации. Любая общая шина является **широковещательной (broadcast)** средой передачи информации. Одно устройство может предавать информацию сразу нескольким или всем остальным устройствам на шине. Заметим, что локальные сети Ethernet и Token Ring с протоколами, использующими общую среду распространения сигналов, также реализуют логический протокол шинной структуры.

4.3.2. Алгоритмы арбитража

1. Алгоритм арбитража на основе статистических приоритетов.

Каждому устройству приписывается уникальный приоритет. Когда несколько устройств одновременно запрашивают шину для передачи, то шина предоставляется устройству с наивысшим приоритетом из числа запросивших. В практической реализации часто используется соединение устройств в цепь, которое фиксирует приоритет каждого устройства местом его подключения к шине, как показано на рис. 4.6. Устройство, расположенное ближе к централизованному блоку управления шиной, имеет больший приоритет.

Запросы выдаются на общую для всех устройств *линию запросов* шины. По принятии сигнала с этой линии централизованный блок управления выдает сигнал на *линию предоставления* шины, если при этом отсутствует сигнал на *линии занятости* шины. Первое из запросивших шину устройств, которое получит сигнал по *линии предоставления* шины, прекращает распространение этого сигнала дальше и информирует блок управления шины выдачей сигнала на линию занятости шины.

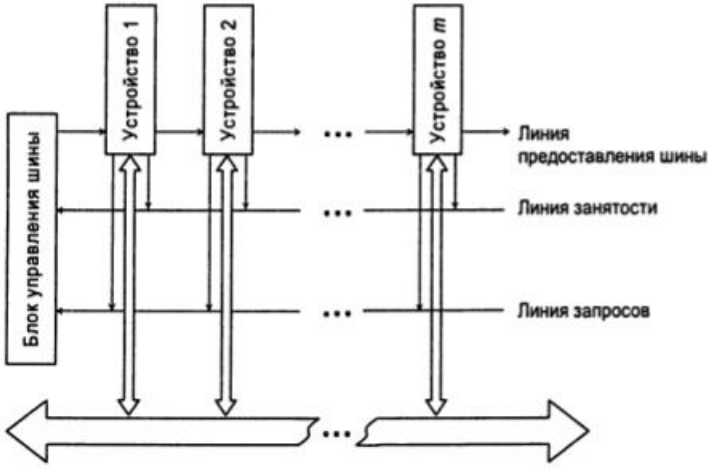


Рис. 4.6. Арбитраж на основе статических приоритетов

2. Алгоритм арбитража на основе фиксированных временных интервалов.

Этот алгоритм предоставляет каждому устройству одинаковый временный интервал по циклической дисциплине. Если устройство получило временной интервал и не имеет данных для передачи, т.е. не использует временной интервал, то этот интервал не предоставляется другому устройству. Алгоритм используется в синхронных шинах, в которых применяется один тактовый генератор для всех устройств.

3. Алгоритм арбитража на основе динамических приоритетов.

Устройствам приписываются уникальные приоритеты, но приоритеты динамически изменяются, предоставляя каждому устройству возможность доступа к шине. Применяются два механизма изменения приоритетов:

1) наивысший приоритет предоставляется устройству, наиболее долго не использовавшему шину (LRU — Least Recently Used);

2) циклическая смена приоритетов (RDC).

Первый механизм переприсваивает приоритет каждого устройства после очередного цикла работы шины.

При RDC-механизме реализуется распределенный блок управления шиной. Линия предоставления шины циклически соединяет все устройства шины, как показано на рис.4.7.

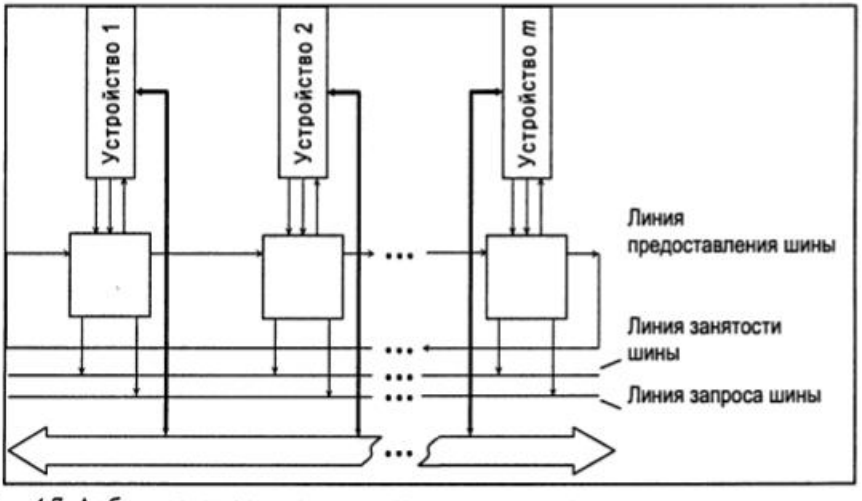


Рис. 4.7. Арбитраж на основе динамических приоритетов

Устройство, получившее доступ к шине, выступает как арбитр шины при следующем цикле арбитража. Приоритет каждого устройства определяется его расстоянием до устройства, выполняющего в текущий момент роль арбитра и имеющего при этом минимальный приоритет.

4. Алгоритм арбитража на основе FIFO-алгоритма.

Механизм основан на ведении очереди запросов к шине по принципу «первым пришел — первым вышел» (FIFO) . Шина предоставляется первому стоящему в очереди устройству. Данный механизм делает шину симметричной, обеспечивая ее максимальную пропускную способность. Однако этот механизм труден в реализации. Во-первых, необходимо организовать очередь достаточной длины и, во-вторых, правильно реализовать арбитраж между почти одновременными запросами.

5. Алгоритм арбитража на основе голосования.

При этом механизме (рис.4.8) линия предоставления шины реализуется как совокупность $\lceil \log_2 m \rceil$ линий голосования, где m — число устройств шины, а $\lceil \log_2 m \rceil$ — ближайшее целое число, большее или равное $\log_2 m$.

При запросе шины арбитр начинает выдавать на линии голосования адреса устройств. При обнаружении устройством своего адреса оно выставляет сигнал на линию занятости. Голосование прекращается с тем, чтобы возобновиться после освобождения шины. Очевидно, что приоритет устройств задается порядком выдачи адресов при голосовании.

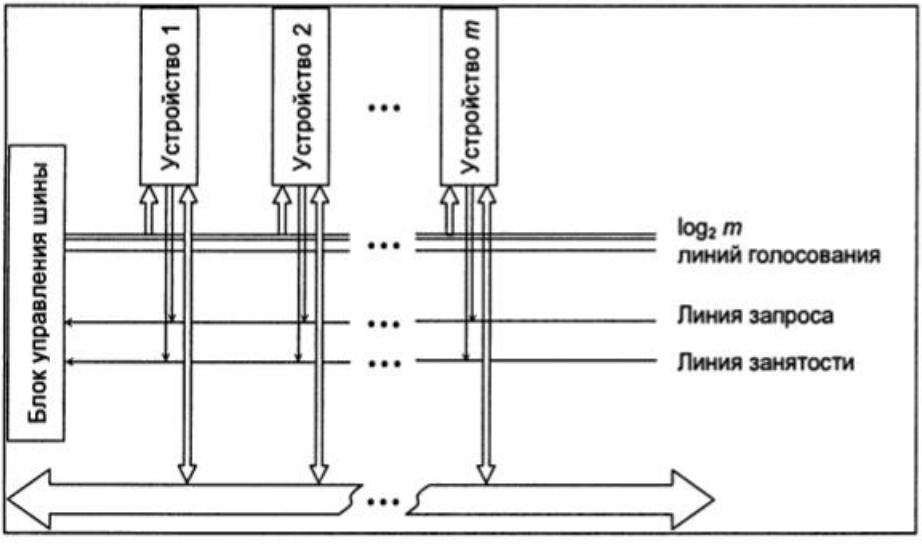


Рис. 4.8. Арбитраж на основе голосования

6. Алгоритм арбитража на основе независимых запросов.

Каждое устройство имеет индивидуальные линии запроса и предоставления шины, как показано на рис.4.9. Примером такой шины может служить PCI.

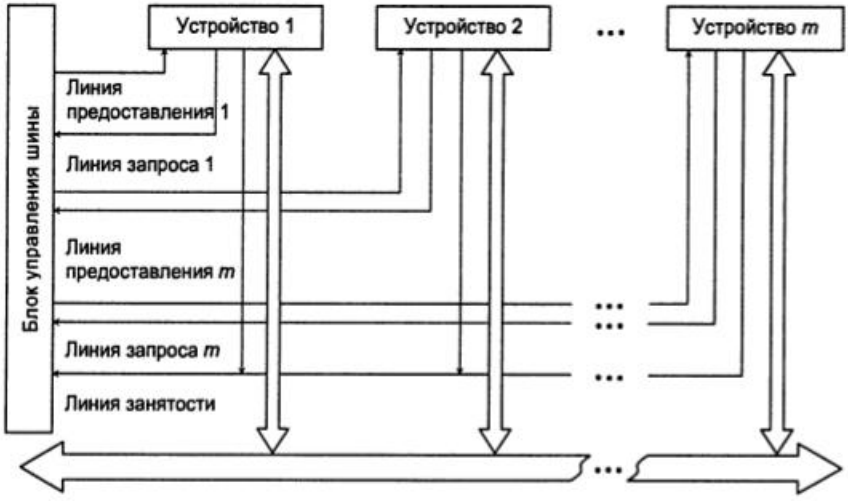


Рис. 4.9. Арбитраж на основе независимых запросов

4.4. СТАНДАРТЫ ШИН

4.4.1. Общие тенденции в развитии шин

Обычно количество и типы устройств ввода-вывода в вычислительных системах не фиксируются, что позволяет конечному пользователю подбирать необходимую конфигурацию. Поэтому стандарты играют огромную роль, позволяя разработчикам ЭВМ и устройств ввода-вывода работать независимо.

Шина есть не что иное, как информационная магистраль. Тот или иной стандарт шины определяет механические (например, конфигурация разъемов), электрические (количество линий шины, величины допустимых напряжений и токов и т. п.) и логические (разрядность, протокол обмена, алгоритм арбитража) характеристики шины.

Как уже было отмечено, шины могут использоваться как для подключения модулей ввода-вывода к центральной части ЭВМ (шины «большого» интерфейса), так и для присоединения к МВВ внешних устройств (шины «малого» интерфейса). Несмотря на различное предназначение, в последнее время можно наблюдать единую тенденцию в развитии шин как «большого», так и «малого» интерфейсов, — переход от параллельных синхронных шин к последовательным асинхронным.

Переход происходит медленно, но неуклонно: сначала клавиатура и мышь, затем модемы, сканеры и принтеры, видеокамеры, цифровые фотоаппараты стали использовать последовательные интерфейсы (например, USB). Оказалось экономически выгоднее перенести максимум функциональности в логику управляющих микросхем (необходимые для последовательной передачи кодирование, передача

и прием, декодирование данных, протоколы маршрутизации и защиты от ошибок и т. п.), нежели иметь дело с многопроводными кабелями, недешевыми пайкой, экранированием, разводкой и избыточными медными контактами.

Со временем тенденция перехода на последовательные шины распространилась на интерфейсы дисковых накопителей (SATA и SAS) и на системные шины (Hyper-Transport и PCI Express). Рассмотрим основные преимущества последовательных шин перед параллельными.

1. Перенос логики практической реализации шины на управляющие микросхемы облегчает отладку, повышает гибкость и сокращает время разработки ЭВМ.
2. Возможность использовать иные носители сигнала, например оптические.
3. Экономия пространства (экономичная миниатюризация) и снижение сложности монтажа.
4. Простота реализации горячих подключений и динамической конфигурации.
5. Возможность выделять гарантированные и изохронные каналы для передачи данных.
6. Лучшая с точки зрения затрат и более гибкая с точки зрения топологии масштабируемость.

В будущем же следует ожидать перехода на беспроводные шины (примером является USB wireless).

4.4.2. Шины «большого» интерфейса

Рассмотрим несколько подробнее общие шины, имеющие (или имевшие в недалеком прошлом) широкое распространение.

Одной из популярных шин персональных компьютеров была системная шина IBM PC/XT, обеспечивавшая передачу 8 бит данных. Кроме того, эта шина включала 20 адресных линий, которые ограничивали адресное пространство пределом в 1 Мбайт. Для работы с внешними устройствами в этой шине были предусмотрены 4 линии аппаратных прерываний (IRQ) и 4 линии для требования внешними устройствами прямого доступа к памяти (DMA). Для подключения плат расширения использовались специальные 62-контактные разъемы. При этом системная шина и микропроцессор синхронизировались от одного тактового генератора с частотой 4.77 МГц. Таким образом, теоретическая скорость передачи данных могла достигать немногим более 4 Мбайт/с.

Системная шина **ISA** (Industry Standard Architecture) впервые стала применяться в персональных компьютерах IBM PC/AT на базе процессора i286. Эта системная шина отличалась наличием второго, 36-контактного дополнительного разъема для соответствующих плат расширения. За счет этого количество адресных линий было увеличено на 4, а данных — на 8, что позволило передавать параллельно 16 бит данных и обращаться к 16 Мбайт системной памяти. Количество линий аппаратных прерываний в этой шине было увеличено до 15, а каналов прямого

доступа — до 7. Системная шина ISA полностью включала в себя возможности старой 8-разрядной шины. Шина ISA позволяла синхронизировать работу процессора и шины с разными тактовыми частотами. Она работала на частоте 8 МГц, что соответствовало максимальной скорости передачи 16 Мбайт/с.

С появлением процессоров i386, i486 и Pentium шина ISA стала узким местом персональных компьютеров на их основе. Новая системная шина **EISA** (Extended Industry Standard Architecture), появившаяся в конце 1988 г., обеспечивала адресное пространство в 4 Гбайт, 32-битовую передачу данных (в том числе и в режиме ПДП), автоматическую конфигурацию системы и плат расширения. Устройства шины ISA могли работать на шине EISA. Шина EISA предусматривала централизованный арбитраж, поэтому к ней могло быть подключено несколько главных устройств. Улучшенная система прерываний позволяла подключать к каждой физической линии запроса на прерывание несколько устройств, что снимало проблему количества линий прерывания. Шина EISA тактировалась частотой около 8 МГц и имела максимальную теоретическую скорость передачи данных 33 Мбайт/с. Шина EISA нашла широкое применение в серверных решениях.

В 1991 г. компания Intel предложила базовую версию (1.0) проекта стандарта шины **PCI** (Peripheral Component Interconnect — «соединение периферийных компонент»), которая была призвана заменить ISA и EISA. В 1993 г. PCI Special Interest Group (PCISIG — Специальная группа интересов PCI) публикует обновленную версию (2.0) стандарта, ставшую основой для широкого внедрения PCI и различных ее модификаций в производство ЭВМ.

В базовой версии (IEEE P1386.1) PCI представляла собой 32-битную синхронную шину с тактовой частотой 33 МГц и максимальной пропускной способностью 133 Мбайт/с. Адресное пространство составляло 32-бита (4 ГБ), использовались сигнальные уровни 3,3 и 5 В.

Помимо увеличенной производительности и масштабируемости, шина PCI обладала двумя принципиальными преимуществами по сравнению с ISA (EISA). Во-первых, PCI не была ориентирована на конкретный микропроцессор (на базе PCI, помимо МП Intel, использовались процессоры Alpha и PowerPC). Во-вторых, стандарт предусматривал автоматическую конфигурацию подключаемых к шине ВУ (так называемая технология «plug-and-play»).

Позже появились следующие ключевые модификации шины PCI:

- PCI 2.2 — допускается ширина шины 64 бит и/или тактовая частота 66 МГц (пиковая пропускная способность до 533 Мбайт/с);
- PCI-X — 64-битная версия PCI 2.2 с увеличенной до 133 МГц частотой (пиковая пропускная полоса 1066 Мбайт/с);
- PCI-X 266 (PCI-X DDR) и PCI-X 533 (PCI-X QDR) — соответственно DDR- и QDR-версии PCI-X. Эффективная частота 266 (533) МГц, реальная 133 МГц с передачей по обоим фронтам тактового сигнала, пиковая пропускная способность 2.1 (4.3) Гбайт/с;
- Mini PCI — PCI для применения в ноутбуках;

• Accelerated Graphics Port (AGP) — высокоскоростная версия PCI, оптимизированная для графических ускорителей. В AGP отсутствует арбитраж шины, поэтому к шине допустимо подключение только одного устройства (за исключением версии AGP 3.0, где устройств и слотов может быть два).

22 июля 2002 г. была опубликована базовая спецификация нового стандарта **PCI Express**, пришедшего на замену PCI. PCI Express представляет собой последовательную системную шину общего назначения, организованную как совокупность независимых самостоятельных последовательных каналов передачи данных. Каждый канал состоит из двух дифференциальных сигнальных пар (уровень напряжения 0,8 В), при этом используется избыточное защищенное от помех кодирование — каждый байт при передаче представляется десятью битами.

Пропускная способность PCI Express составляет 2,5 Гбит/с для одного канала в каждом направлении одновременно. Стандартизированы 1-, 2-, 4-, 8-, 16- и 32-канальные варианты. При передаче данные передаются параллельно (но не синхронно) по всем доступным каналам.

Вся контрольная информация передается по тем же линиям что и данные, используется стек протоколов из нескольких уровней, включая маршрутизацию данных. Стандарт предусматривает использование альтернативных носителей сигнала, таких, как оптические волноводы.

Шина **VME** приобрела большую популярность как шина ввода-вывода в рабочих станциях и серверах на базе RISC-процессоров. Имеются несколько версий этого стандарта. В частности, VME32 — 32-битовая шина с производительностью 30 Мбайт/с, а VME64 — 64-битовая шина с производительностью 160 Мбайт/с.

В однопроцессорных и многопроцессорных рабочих станциях и серверах на основе микропроцессоров SPARC одновременно используются несколько типов шин: SBus, MBus и XDBus, причем шина SBus применяется в качестве шины ввода-вывода, а MBus и XDBus — в качестве шин для объединения большого числа процессоров и памяти.

Заметим, что в современных ВС часто применяются и специализированные (запатентованные) шины, обеспечивающие очень высокую пропускную способность для построения многопроцессорных серверов. Одним из таких решений является системная шина **POWERpath-2**, которая применяется в SMP-системе Power Challenge компании Silicon Graphics. Она способна поддерживать эффективную работу до 36 процессоров MIPS R4400 (9 процессорных плат с четырьмя 150 МГц процессорами на каждой плате) с общей расслоенной памятью объемом до 16 Гбайт (коэффициент расслоения памяти равен восьми). POWERpath-2 имеет разрядность данных 256 бит, разрядность адреса 40 бит, и работает на частоте 50 МГц с пониженным напряжением питания. Шина поддерживает методику расщепления транзакций, причем может иметь до восьми отложенных транзакций чтения одновременно. При этом арбитраж шины адреса и шины данных выполняется независимо. POWERpath 2 поддерживает протокол когерентного состояния кэш-памяти каждого процессора в системе.

4.4.3. Шины «малого» интерфейса

Рассмотрим старейший стандарт **IDE** (Integrated Drive Electronics), или **ATA** (Advanced Technology Attachment), который был разработан в 1989 г. тремя компаниями (CDC (подразделение Imprimus), Western Digital и Compaq) для подключения накопителей на жестких магнитных дисках (НЖМД) к ПЭВМ. Официально стандарт под именем ATA был утвержден в 1994 г. комитетом T13 Национального комитета по стандартам информационных технологий (NCITS). Стандарт был определен как 16-битный шинный интерфейс для обмена информацией между НЖМД и контроллерами на системной шине. Заметим, что ATA — официальное название стандарта, в то время как аббревиатура IDE используется в маркетинговых целях.

В 1999 г. стандарт ATA был заменен на стандарт ATAPI (ATA Packet Interface) для возможности подключения дисководов CD-ROM и стримеров в качестве IDE-устройств.

В течение почти двух десятилетий стандарт ATA постоянно обновлялся (от версии 1 до версии 6), сохраняя обратную совместимость устройств. Улучшения касались в основном разрядности и режима прямого доступа к памяти, скорость передачи данных в котором постоянно увеличивалась. Например, стандарт **ATAPI-6** предписывает разрядность 64 бита и скорость передачи данных в режиме ПДП (в стандарте режим называется UltraDMA) до 133 Мбайт/с. Кроме этого, в стандарт вводились новые команды, рассчитанные на передачу аудио/видео потоков, и реализовывались методы снижения уровня шума НЖМД.

В 2000-х годах был осуществлен переход на стандарт **Serial ATA** (SATA), который, являясь логически совместимым с IDE, предусматривает не параллельную, а последовательную передачу данных. Работа SATA в полудуплексном режиме позволяет достичь пропускной способности до 1,5 Гбайт/с.

Другой популярной шиной «малого» интерфейса вплоть до настоящего времени является шина **SCSI**. Под термином **SCSI** — Small Computer System Interface («интерфейс малых вычислительных систем», произносится «скази») обычно понимается набор стандартов, разработанных Национальным институтом стандартов США (ANSI) и определяющих механизм реализации магистрали передачи данных между системной шиной ЭВМ и внешними устройствами. В отличие от ATA, SCSI изначально был ориентирован на возможность подключения к одному модулю (контроллеру) ввода-вывода нескольких внешних устройств различных типов.

Автором стандарта считается М. Шугарт, на основании разработок которого в 1986 г. был принят начальный стандарт X3.131, известный теперь под названием SCSI-1. В этом стандарте можно было к одному МВВ подключить до восьми внешних устройств, включая контроллер. Интерфейс содержал развитые средства управления и в то же время не был ориентирован на какой-либо конкретный тип ВУ. Шина данных имела 8 разрядов, максимальная скорость передачи — до 1,5 Мбайт/с в асинхронном режиме (по методу «запрос-подтверждение»), и до 5 Мбайт/с в синхронном режиме (метод «несколько запросов — несколько под-

тверждений»). Для обнаружения ошибок использовался контроль четности. Электрически стандарт был реализован в виде 24 линий.

SCSI-2 (1989) представлял собой существенное развитие базового SCSI. В режиме Fast SCSI была увеличена скорость передачи (до 3 Мбайт/с в асинхронном и до 10 Мбайт/с в синхронном режиме). Были добавлены новые команды и сообщения, поддержка контроля четности сделана обязательной. В режиме Wide SCSI была введена возможность расширения шины данных до 16 разрядов, что обеспечило скорость до 20 Мбайт/с. Кроме этого, был разработан новый 68-контактный соединительный разъем.

Последующая спецификация, SCSI-3 (1992), уже не только ввела новые скорости передачи, но и значительно расширила систему команд. Кроме того, в качестве среды передачи допускалось использование, наряду с традиционным параллельным шинным интерфейсом, и других параллельных и последовательных протоколов: Fibre Channel, IEEE 1394 Firewire и Serial Storage Protocol (SSP).

Интерфейс Ultra-2 SCSI (1997), использовал частоту шины 20 МГц. Интерфейс Ultra/Wide SCSI поддерживал 16 устройств и обеспечивал скорость передачи данных до 40 Мбайт/с. Следующие интерфейсы — Ultra-3 SCSI (1999), Ultra 320 SCSI, Ultra 640 SCSI (2003) — не привнесли ничего принципиально нового в стандарт, кроме скорости (160, 320 и 640 Мбайт/с соответственно). Они остаются также с шириной шины 16 бит, и также к интерфейсу можно подключить до 16 внешних устройств.

К настоящему времени стандарт SCSI исчерпал свои возможности. Логичным и естественным последовательным расширением технологии параллельного интерфейса SCSI является стандарт последовательного интерфейса Serial-Attached SCSI (SAS).

Стандарт **SAS** опирается на электрические и физические характеристики интерфейса Serial ATA, обеспечивает масштабируемость, производительность, надежность и управляемость данных в серверах и подсистемах хранения данных. Архитектурная схожесть с SATA не мешает SAS обладать наиболее востребованными чертами SCSI.

SAS — это полнодуплексный SATA с поддержкой двух портов, больших возможностей адресации, высокой надежностью, производительностью и логической совместимостью со SCSI. Интерфейс Serial ATA можно рассматривать как упрощенное подмножество Serial Attached SCSI для работы в простых системах без критических требований к надежности и производительности.

В отличие от SCSI, SAS имеет последовательную архитектуру с непосредственным подключением контроллера к ВУ. SAS поддерживает до 128 ВУ различных типов, совместно подключенных более тонкими и длинными, нежели в случае SCSI, кабелями. Полнодуплексный сигнальный последовательный интерфейс SAS обеспечивает обмен данными на скорости до 3,0 Гбайт/с на порт.

В заключение рассмотрим широко распространенную универсальную последовательную шину **USB** (Universal Serial Bus).

Открытый, доступный всем производителям оборудования и программного обеспечения, стандарт USB был разработан в средине 90-х годов прошлого века

группой компаний: IBM, DEC, Intel, Microsoft, NEC и Northern Telecom. Изначально в стандарт USB было заложено несколько удачных принципов.

1. Универсальный интерфейс, по которому может быть подключено любое из устройств: принтер, модем, клавиатура, мышь, аудиосистема, флэш-накопитель, а в последнее время еще и CD-ROM или жесткий диск. Определенные классы подключаемых устройств могут получать питание от встроенного в USB источника 5 В.

2. Горячее подключение ВУ, не требующее установки дополнительных плат, отдельных драйверов и выключения ЭВМ.

3. Возможность каскадного подключения: некоторые устройства могут выступать как узлы (концентраторы), к которым можно подключить дополнительные устройства.

Стандарт **USB 1.1** поддерживает две скорости обмена — высокую (до 12 Мбит/с, длина кабеля до 3 м) и низкую (1,5 Мбит/с, длина кабеля до 5 м). Максимальное количество подключаемых ВУ (включая концентраторы) равно 127, при этом возможно одновременное подключение ВУ с разными скоростями обмена. Стандарт USB 2.0 устанавливает максимальную скорость до 60 Мбайт/с. На уровне программного обеспечения USB поддерживается практически любой операционной системой.

USB OTG (от On The Go — «на ходу») — дальнейшее расширение спецификации USB 2.0, предназначенное для соединения периферийных USB-устройств друг с другом без необходимости подключения к ЭВМ. Например, цифровую камеру можно подключить к фотопринтеру напрямую, если они оба поддерживают стандарт USB OTG.

USB wireless позволяет организовать беспроводную связь с высокой скоростью передачи информации (до 480 Мбит/с на расстоянии 3 м и до 110 Мбит/с на расстоянии 10 м).

4.5. УСТРОЙСТВА ВВОДА-ВЫВОДА

4.5.1. Основные типы устройства ввода-вывода

Как правило, периферийные устройства ЭВМ делятся на устройства ввода, устройства вывода и внешние запоминающие устройства, осуществляющие как ввод данных в машину, так и вывод данных из компьютера. Основной обобщающей характеристикой устройств ввода-вывода может служить скорость передачи данных (максимальная скорость, с которой данные могут передаваться между устройством ввода-вывода и основной памятью или процессором).

Таблица 4.1

Основные характеристики устройств ввода-вывода

Тип устройства	*Направление передачи данных*	*Скорость передачи данных*
Клавиатура	Ввод	0.01 Кбайт/с
Мышь	Ввод	0.02 Кбайт/с

Голосовой ввод	Ввод	0.02 Кбайт/с
Сканер	Ввод	200.0 Кбайт/с
Голосовой вывод	Вывод	0.06 Кбайт/с
Строчный принтер	Вывод	1.00 Кбайт/с
Лазерный принтер	Вывод	100.0 Кбайт/с
Графический дисплей	Вывод	Сотни Мбайт/с
Оптический диск	ЗУ	Десятки Мбайт/с
Магнитная лента	ЗУ	Десятки Мбайт/с
Магнитный диск	ЗУ	Сотни Мбайт/с

В табл. 4.1 представлены основные устройства ввода-вывода, применяемые в современных ЭВМ, а также указаны примерные скорости обмена данными, обеспечиваемые этими устройствами.

Рассмотрим более подробно ЗУ и ряд перспективных УВВ.

4.5.2. Запоминающие устройства

Дисковые накопители являются самыми распространенными внешними запоминающими устройствами, чему способствует оптимальное сочетание емкости, стоимости и скорости работы. Все дисковые накопители представляют собой внешнюю память с прямым доступом. В современных ЭВМ в основном используются накопители **на жестких магнитных дисках (НЖМД)** и **на оптических дисках (НОД)**. Еще совсем недавно широкое применение находили накопители **на гибких магнитных дисках (НГМД, дискеты)** и **магнитооптические диски (МОД)**.

4.5.2.1. Накопители на жестких магнитных дисках

Накопитель на жестких магнитных дисках, или просто **жесткий диск (Hard Disk Drive — HDD),** обычно состоит из набора пластин, представляющих собой металлические диски, покрытые магнитным материалом и соединенные между собой при помощи центрального шпинделя (рис. 4.10). Для записи данных используются обе поверхности пластины. В современных дисковых накопителях используется от 1 до 9 пластин. Шпиндель вращается с высокой постоянной скоростью (в современных дисках 5400 или 7200 оборотов в минуту). Каждая пластина содержит набор концентрических записываемых дорожек. Обычно дорожки делятся на блоки данных объемом 512 байт, часто называемые секторами. Количество блоков, записываемых на одну дорожку, зависит от физических размеров пластины и плотности записи.

Данные записываются или считываются с пластин с помощью **головок записи/считывания**, по одной на каждую поверхность. Головки объединяются в блок, который с помощью тонармов соединен со шпинделем двигателя. Двигатель представляет собой электромеханическое устройство, которое, вращая шпиндель, позиционирует блок головок над заданным цилиндром. **Цилиндр** — это набор дорожек, соответствующих одному положению блока головок. Таким образом,

накопитель на жестких магнитных дисках представляет собой набор пластин, магнитных головок, тонармов, двигателя плюс воздухонепроницаемый корпус. Кроме этого, НЖМД содержит электронные схемы управления, реализованные в виде миниатюрной печатной платы, и стандартные разъемы для подсоединения питания и шины «малого» интерфейса.

Рис. 4.10. Схематичный вид жесткого диска без корпуса

Процесс обращения (чтения или записи) к жесткому диску показан на рис. 4.11. Он включает в себя 3 этапа: перемещение блока головок чтения/записи на нужную дорожку *а*), ожидание подхода требуемого сектора под головки чтения/записи *б*) и собственно передача данных, считываемых с диска или записываемых на него *в*). Каждый из этих этапов занимает определенное время, входящее в общее время обращения к диску. Все этапы так или иначе связаны с механическими перемещениями, поэтому их времена сравнительно велики и составляют величины порядка единиц миллисекунд.

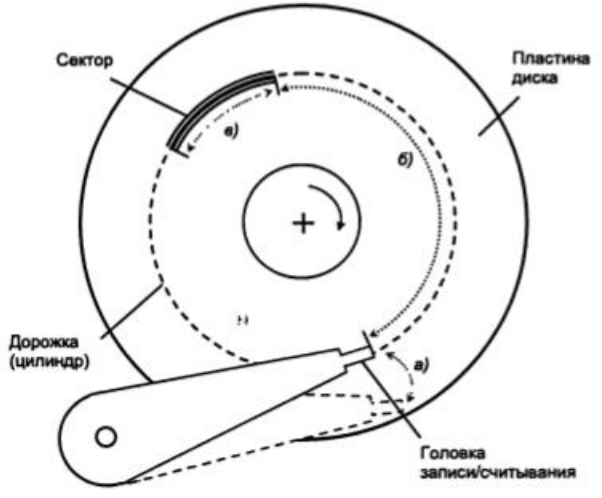


Рис. 4.11. Обращение к жесткому диску

Таким образом, производительность диска является функцией времени обслуживания, которое включает три основных компонента: время доступа, время ожидания и время передачи данных.

Время доступа — это время, необходимое для позиционирования головок на соответствующую дорожку, содержащую искомые данные. Оно является функцией затрат на начальные действия по ускорению головки диска, а также функцией числа дорожек, которые необходимо пересечь на пути к искомой дорожке. Вторым компонентом времени обслуживания является **время ожидания**. Пластина диска должна повернуться, чтобы требуемый сектор был совмещен с положением головки. После этого данные могут быть записаны или считаны. Время ожидания напрямую зависит от скорости вращения пластин.

Последним компонентом является **время передачи данных**, т. е. время, необходимое для физических процессов считывания и передачи информации. Время передачи данных является функцией от числа передаваемых байтов (размера блока), скорости вращения, плотности записи на дорожке и скорости срабатывания электронных схем.

К ЭВМ жесткие диски подключаются с помощью шины «малого» интерфейса. В настоящее время для этого широко используются современные последовательные шины Serial ATA, Serial Attached SCSI и USB. На момент написания учебника в эксплуатации еще находится значительное число ЭВМ с жесткими дисками, подключенными по параллельным интерфейсам: ATA (IDE) и SCSI.

Значение жесткого диска в составе ЭВМ трудно переоценить. Фактически НЖМД является основным энергонезависимым, но в то же время оперативным и быстрым хранилищем данных. Кроме этого, пространство жестких дисков непосредственно включается в иерархическую структуру памяти ЭВМ и широко используется для организации виртуальной памяти. За счет дискового пространства объем используемой виртуальной памяти может значительно превышать емкость основной физической памяти, установленной в ЭВМ. Это позволяет, с одной стороны, одновременно работать с большим множеством программ, с другой стороны, обрабатывать в одной программе данные объемом, превышающим размер физической памяти.

По быстродействию жесткие диски уступают только оперативной памяти (хотя отставание здесь составляет 3–4 порядка) за счет эффективной организации **прямого доступа** к хранимой информации. Напомним, что при прямом доступе информация хранится уникально идентифицируемыми блоками. Время доступа к блоку не зависит от того, какое место он занимает в последовательности других блоков, но зависит от его физического местоположения. После доступа к блоку информация из него считывается последовательно.

В жестких дисках роль блока играет сектор, для идентификации которого используется следующая информация:

• номер цилиндра, содержащего сектор;

• номер головки (пластины) в цилиндре — однозначно определяет содержащую сектор дорожку;

• номер сектора на дорожке.

В жестких дисках недавнего прошлого идентификатор сектора соответствовал его физическому местоположению, т. е. номера цилиндра, головки и сектора на дорожке определялись по реальным параметрам жесткого диска. Физические секторы в процессе эксплуатации могут повреждаться, что влечет за собой проблемы в размещении информации (использовать поврежденный сектор нельзя), решение которых возлагалось на операционную систему. Со временем электроника НЖМД приобрела возможность автоматически определять и исключать из работы сбойные секторы, и идентификация сектора по его физическому расположению стала неудобной.

В современных дисковых накопителях идентификатор сектора является логическим, так как диск представляется для ЭВМ как совокупность некоторого числа виртуальных пластин, каждая из которых содержит некоторое число виртуальных дорожек, состоящих, в свою очередь, из некоторого числа виртуальных секторов. Например, жесткий диск физически может состоять из двух двусторонних пластин, но для ЭВМ представляться как 9-пластинное устройство. В этом случае отображение логического идентификатора сектора в его физические параметры производится электронными схемами НЖМД, которые способны в случае обнаружения сбоя автоматически заменить сбойный физический сектор на исправный без смены логического идентификатора.

Накопители на жестких магнитных дисках могут иметь различные уровни сменяемости. Обычный накопитель на жестком диске (*винчестер*) устанавливается на компьютер надолго, и для его подключения требуется открыть корпус системного блока и отключить питание. Существуют и накопители, допускающие *«горячую замену» (hot swap)*, для которой не требуется отключение питания и имеется специальный конструктив, позволяющий устанавливать и снимать винчестер со стороны лицевой панели, не разбирая компьютер. Однако такая замена производится в основном в сервисных целях. Съемные винчестеры — устройства с аналогичными свойствами, но предназначенные для переноса или обеспечения безопасного (в смысле конфиденциальности) хранения данных. Выпускаются недорогие переходники Mobile Rack, позволяющие использовать в качестве съемного обычный ATA-диск. При этом накопитель 3,5″ устанавливается во внешний отсек 5″. Однако следует помнить, что обычные накопители не рассчитаны на вибрацию и удары, опасность которых при частой переноске устройств повышается. Кроме этого, разъемы недорогих переходников не рассчитаны на интенсивное использование (частое соединение/разъединение).

4.5.2.2. Накопители на гибких магнитных дисках и магнитооптические накопители

В настоящее время практически прекращен массовый выпуск этих видов дисковых накопителей, хотя в эксплуатации еще находится значительное число ЭВМ, оборудованных данными ЗУ.

Запись на **магнитооптические диски (МО-диски)** выполняется при взаимодействии лазера и магнитной головки. Луч лазера разогревает до точки Кюри (температуры потери материалом магнитных свойств) микроскопическую область записывающего слоя, которая при выходе из зоны действия лазера остывает, фиксируя магнитное поле, наведенное магнитной головкой. Для МО-дисков данные, записанные на диск, выдерживают сильные магнитные поля и колебания температуры. Все функциональные свойства дисков сохраняются в диапазоне температур от –20 до +50 °С.

МО-диски уступают обычным жестким магнитным дискам лишь по времени доступа к данным. В магнитооптическом дисководе используются сменные диски, что обеспечивает практически неограниченную емкость. Хотя стоимость хранения единицы данных на МО-дисках меньше стоимости хранения того же объема данных на жестких магнитных дисках, МО-диски были вытеснены с рынка еще более дешевыми функциональными оптическими дисками (CD и DVD).

Накопители на гибких магнитных дисках (НГМД, дискеты) в классическом исполнении остановились на емкости носителя в 2,88 Мбайт, но самое широкое распространение получили 3,5-дюймовые дискеты емкостью 1,44 Мбайт. **НГМД (FDD — Floppy Disk Drive)** были разработаны сотрудником фирмы IBM Аланом Шугартом в конце 60-х годов. Гибкий диск (дискета) схож с жестким диском. Дискета диаметром 85 мм (3,5″) имеет 80 концентрических дорожек с обеих сторон, на которых могут быть записаны по 9, 18 или 36 секторов по 512 байт каждый (соответственно емкость дискеты будет равна 720 Кбайт, 1,44 Мбайт и 2,88 Мбайт). Начало дорожки на 3,5″ дискетах отличается специальным индексным отверстием, которое расположено в металлической вставке, занимающей центральную часть дискеты. Привод такого накопителя включает в себя электромеханическую часть с блоком головок чтения/записи и электронную часть. Угловая скорость вращения дискеты 300–360 об/мин. Плотность записи информации на дискету составляет 135 дорожек/дюйм. Для повышения емкости требовалось радикальное увеличение плотности записи, в первую очередь за счет увеличения числа дорожек. Однако при этом требовалось точное позиционирование, что на обычном шаговом двигателе (разомкнутая система) недостижимо.

Начиная с 80-х годов прошлого века, в течение 25 лет дискеты фактически являлись основным видом съемных носителей практически во всех видах ЭВМ. Более того, процесс инициализации ЭВМ — первая загрузка для установки ОС — производился долгое время исключительно с НГМД. Со временем невозможность увеличения емкости дискет, с одной стороны, и крайне низкая их надежность (по

сравнению с оптическими дисками и флэш-памятью) — с другой, сделали 3,5″ дискеты достоянием истории.

Накопители **Zip** были представлены фирмой **Iomega** в 1994 г. как альтернатива стандартным дискетам. Накопители Zip широко использовались как устройства архивации и переноса данных. Благодаря невысокой цене и значительной по сравнению с дискетами емкостью они получили широкое распространение.

Накопители Zip (Zip-диски) представляют собой сменные носители информации емкости 100 и 250 Мбайт. Для чтения и записи Zip-дисков используется отдельное внешнее устройство, которое, как правило, подключается к ЭВМ либо через шины IDE, SCSI, либо через порт принтера. Пластина сменного диска размещается в специальном картридже 3,5″. Скорость вращения диска — 2941 об/мин. В исходном положении головки выведены из картриджа, слегка разведены и лежат на очищающей площадке. В рабочем состоянии головки вводятся в картридж с торца и зависают над поверхностью вращающегося диска, не касаясь ее. Носители чувствительны к термодеформации — диск, записанный в горячем состоянии, будет плохо читаться в холодном.

4.5.2.3. Дисковые массивы и уровни RAID

Одним из способов повышения производительности дисковой подсистемы является использование параллелизма путем объединения нескольких физических дисков в матрицу (группу) с организацией их работы аналогично одному логическому диску. К сожалению, надежность матрицы любых устройств падает при увеличении числа устройств. Полагая интенсивность отказов постоянной, т. е. при экспоненциальном законе распределения наработки на отказ, а также при условии, что отказы независимы, получим, что **среднее время безотказной работы (Mean Time To Failure — MTTF)** матрицы дисков будет равно частному от деления:

$$\text{MTTF} = \frac{\text{MTTF одного диска}}{\text{Число дисков в матрице}}.$$

Для достижения повышенного уровня отказоустойчивости приходится использовать дополнительные диски с избыточной информацией, позволяющей восстановить исходные данные при отказе диска. Подобные избыточные матрицы получили название **RAID — избыточный массив независимых дисков (Redundant Array of Independent Disks)**. Существуют несколько способов объединения дисков RAID. Каждый уровень (всего их 8) представляет свой компромисс между пропускной способностью ввода-вывода и емкостью диска, предназначенной для хранения избыточной информации. Впервые термин RAID появился в 1987 г., когда исследователям из Калифорнийского университета в Беркли удалось создать действующий массив из нескольких жестких дисков.

Если происходит отказ какого-либо диска, то предполагается, что в течение короткого интервала времени он будет заменен и информация будет восстановлена

на новом диске с использованием избыточной информации. Это время называется **средним временем восстановления (Mean Time To Repair — MTTR) $T_{вср}$**. Этот показатель можно уменьшить, если в систему входят дополнительные диски в качестве «горячего резерва»: при отказе диска резервный диск подключается аппаратно-программными средствами. Периодически оператор вручную заменяет все отказавшие диски. Процесс замены отказавшего диска включает следующие этапы:

- определение отказавшего диска;
- устранение отказа без останова обработки;
- восстановление потерянных данных на резервном диске;
- периодическая замена отказавших дисков на новые.

Повышение производительности в RAID-системе достигается за счет применения **расслоения (striping)**, подобного чередованию адресов в оперативной памяти. Дисковое пространство разбивается на блоки — так называемые **полосы** (см. рис. 4.12), которые распределяются по различным дискам массива. Если полосы расположены на разных дисках, возможно их параллельное считывание.

Чаще всего логически последовательно идущие друг за другом полосы распределяются по дискам *n*-дискового массива следующим образом. Первые *n* логических полос физически располагаются как первые полосы на каждом из *n* дисков, следующие *n* полос — как вторые полосы на каждом физическом диске и т. д. Набор логически последовательных полос, одинаково расположенных на каждом диске массива, называется **поясом** (см. рис. 4.12).

Рис. 4.12. RAID уровня 0

Рассмотрим различные уровни RAID. Следует отметить, что в отличие от привычного понятия иерархии в уровнях RAID нет преемственности «снизу вверх», т. е. RAID3 не является улучшенной версией RAID2, а RAID5 — улучшенной версией RAID1. Поэтому в настоящее время используют как простые (single) так и составные (multiple) RAID-массивы. Составные RAID-массивы являются сочетанием двух простых.

RAID0: дисковый массив без отказоустойчивости.

Этот вид RAID массива (см. рис. 4.12) является простейшим и базируется на расслоении дискового пространства. Вся входящая информация разбивается на блоки фиксированной длины и записывается на все имеющиеся накопители. При наличии от 2 до 4 дисков RAID0 дает существенный выигрыш в производительности. Этот вид RAID-массива обеспечивает:

- максимальную эффективность использования дискового пространства;
- простоту реализации;
- низкую стоимость;
- достаточно высокую производительность при решении задач, требующих интенсивной обработки запросов ввода-вывода и данных большого объема.

Недостатком является крайне низкая надежность — отказ одного из дисков влечет за собой потерю всего массива. Поэтому RAID0 применяется там, где важна производительность, но не критична потеря данных. Примером эффективного использования RAID0 может служить хранилище данных для быстрого чтения (видеотека, фотобанк), при этом имеется копия данных (архив) на другом носителе (например, магнитной ленте).

RAID1: зеркальные диски.

Традиционным способом повышения надежности магнитных дисков является использование зеркальных дисков (см. рис. 4.13). Это наиболее дорогостоящий из рассматриваемых способов, так как все диски дублируются, и при каждой записи информация записывается также и на проверочный диск.

Запрос на чтение обслуживается тем из двух дисков, которому в данный момент требуется меньшее время поиска. Запрос на запись обновляет обе соответствующие полосы. При этом отсутствуют временные затраты на вычисление соответствующей корректирующей информации.

Рис. 4.13. RAID уровня 1

Дублирование всех дисков фактически означает удвоение стоимости всей системы или, иначе, использование лишь 50% емкости диска для хранения данных.

Повышение емкости, на которое приходится идти, составляет 100%. Такая низкая экономичность привела к появлению следующего уровня RAID— RAID2.

RAID2: матрица с поразрядным расслоением.

Один из путей достижения надежности при снижении потерь емкости дискового пространства заключается в использовании избыточных контрольных разрядов для исправления одиночных и обнаружения двойных ошибок. В этом методе поток данных разбивается **на слова** таким образом, что количество бит в слове равно количеству дисков и при записи слова каждый отдельный бит записывается на свой диск. В качестве проверочных кодов используются коды Хэмминга. Для каждого слова вычисляется код коррекции ошибок, который записывается на выделенные диски для хранения контрольной информации. Их число равно количеству бит в контрольной сумме. Например (см. рис.4.14), слово *a* состоит из 4 бит $a_0 - a_3$, каждый из которых размещается на отдельном диске (4 диска с информацией $D_0 - D_3$). В этом случае корректирующий код Хэмминга будет составлять три бита $f_i(a)$, $i \in \{0, 1, 2\}$, и под контрольную информацию отводятся три диска $D_4 - D_6$. Каждый бит кода Хэмминга для одного слова информации записывается на отдельный диск.

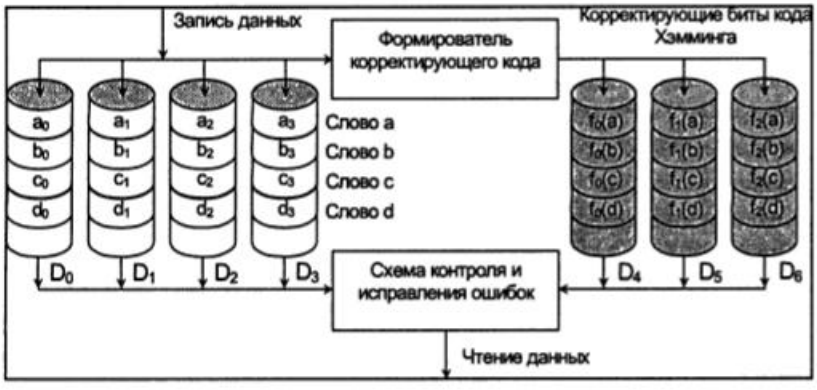


Рис. 4.14. RAID уровня 2

Обычно шпиндели всех дисков синхронизированы так, что головки каждого диска в каждый момент времени находятся в одинаковых позициях. Такая организация обеспечивает лишь один поток ввода-вывода для каждого слова независимо от его размера. Слова большого размера приводят к снижению избыточной емкости, идущей на обеспечение отказоустойчивости. Например, при использовании 32-разрядного слова корректирующий код будет состоять из 7 бит, но такой массив, очевидно, будет существенно дороже.

При записи больших массивов данных системы уровня 2 имеют такую же производительность, что и системы уровня 1, но в них используется меньше контрольных дисков, и в этом они превосходят системы уровня 1.

Очевидно, что как информационные биты, так и биты кода Хэмминга при записи на диски объединяются в блоки, размером минимум в один сектор диска. При изменении хотя бы одного бита в блоке (что неизбежно, например, при изменении одного слова) весь блок должен быть перезаписан. Поэтому при передаче небольших порций данных (не превышающих размер блока) производительность RAID2 теряется.

В современных дисковых накопителях, как правило, имеется блок вычисления корректирующих кодов и исправления ошибок. В таких условиях использование нескольких избыточных дисков является неэффективным. По этой причине массивы RAID2 в настоящее время не используются.

RAID3: аппаратное обнаружение ошибок и четность.

Уровень RAID3 (см. рис.4.15) предполагает аппаратное обнаружение ошибок и контроль четности. В этом методе данные записываются на несколько дисков байтами, и один диск выделяется для хранения контрольных сумм, вычисляемых при записи данных. Поломка любого из дисков массива не приводит к потере информации, так как ее можно восстановить при помощи операции «исключающее ИЛИ», примененной к информации на оставшихся дисках.

Например, массив из 5 дисков использует диски $D_0 - D_3$ — для хранения данных, а диск D_4 — для хранения контрольной суммы, вычисляемой для i-го бита как

$$D4_i = D0_i \oplus D1_i \oplus D2_i \oplus D3_i.$$

Пусть отказал диск D_2. Тогда для восстановления информации с этого диска можно использовать контрольную сумму, хранящуюся на диске D_4:

$$D2_i = D0_i \oplus D1_i \oplus D4_i \oplus D3_i.$$

Рис. 4.15. RAID уровня 3

RAID-массивы уровня 3 позволяют достичь высоких скоростей передачи данных за счет использования маленьких полос данных. Каждый запрос на чтение-запись приводит к параллельной передаче данных со всех дисков.

RAID4: внутренний параллелизм.

Этот метод во многом схож с RAID3, но поток данных разделяется не на уровне байтов, а на уровне **блоков**, каждый из которых записывается на отдельный диск (рис. 4.16). После записи группы блоков вычисляется контрольная сумма, которая записывается на отдельный диск.

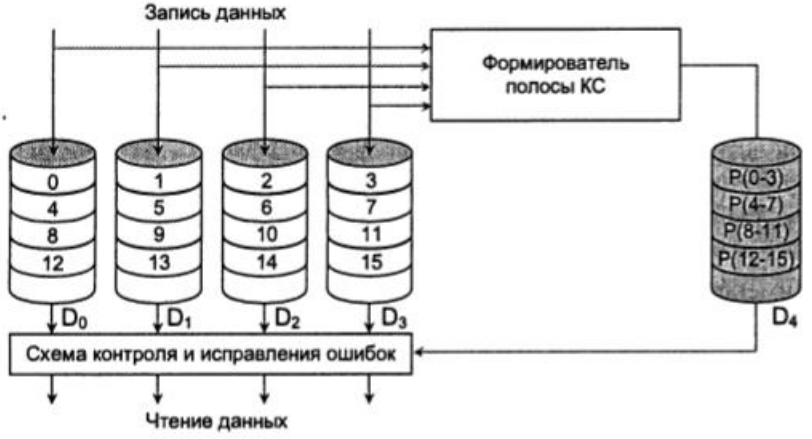


Рис. 4.16. RAID уровня 4

Каждый раз для выполнения записи необходимо обновить не только сами данные, но и соответствующие контрольные суммы. Пусть производится запись в полосу диска D_2. Перед записью для каждого i-го бита мы имеем следующее соотношение:

$$D4_i = D0_i \oplus D1_i \oplus D2_i \oplus D3_i.$$

Обозначим с помощью апострофа потенциально измененные биты. Тогда

$$\begin{aligned} D4'_i &= D0_i \oplus D1_i \oplus D2'_i \oplus D3_i = \\ &= D0_i \oplus D1_i \oplus D2_i \oplus D3_i \oplus D2_i \oplus D2'_i = \\ &= D4_i \oplus D2_i \oplus D2'_i. \end{aligned}$$

Как видим, при каждой записи необходимо прочитать старую полосу данных и старую полосу контрольных сумм. Затем заменить их на новые полосы данных и контрольных сумм. Таким образом, запись каждой полосы связана с двумя операциями чтения и двумя операциями записи. Поэтому существенный выигрыш в производительности RAID4 дает только при записи большого объема информации, охватывающего полосы на всех дисках. В этом случае нет необходимости считывать старые данные для вычисления контрольных сумм, в то же время запись будет производиться параллельно на все диски.

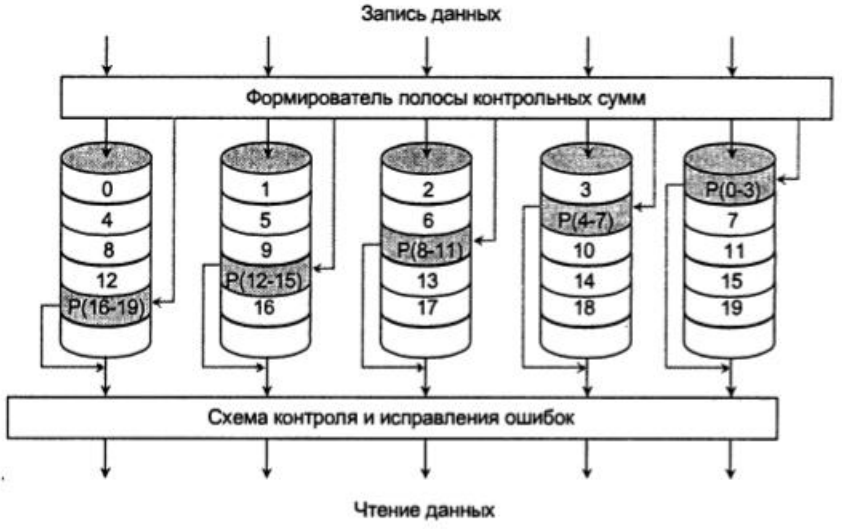


Рис. 4.17. RAID уровня 5

RAID5: распараллеливание записей.

Система уровня 5 (рис. 4.17) улучшает возможности системы уровня 4 посредством распределения контрольной информации между всеми дисками группы. В этом методе **блоки данных и контрольные суммы** циклически записываются на все диски массива, отсутствует выделенный диск для хранения информации о четности (на рис. 4.17 блоки контрольных сумм показаны серым цветом).

Это небольшое изменение оказывает огромное влияние на производительность записи небольших массивов информации. Если операции записи могут быть спланированы так, чтобы обращаться за данными и соответствующими им блоками четности к разным дискам, появляется возможность параллельного выполнения $\frac{N}{2}$ записей, где N — число дисков. Данная организация имеет одинаково высокую производительность при записи и при считывании как небольших, так и больших объемов информации, что делает ее наиболее привлекательной в случаях смешанных применений.

RAID6: двумерная четность.

Как и в RAID5, данные разбиваются на полосы размером в блок, вычисляются полосы контрольных сумм. Полосы данных и контрольных сумм распределяются по всем дискам массива. Доступ к полосам независимый и асинхронный. Различие состоит в том, что на каждом диске хранится не одна, а две полосы контрольных сумм. Первая, как и в RAID5, содержит контрольную сумму для полос, расположенных в горизонтальном срезе массива. В дополнение формируется и записывается вторая полоса, содержащая контрольные суммы для всех полос одного определенного диска, но только не того, на который записывается сама полоса контрольных сумм.

Рассмотрим работу массива RAID6 на примере рис. 4.18. Серым цветом выделены блоки контрольных сумм. При этом сумма $P(0)$ будет сформирована информационными блоками A_0, B_0, C_0; сумма $P(1)$ — блоками A_1, B_1, C_1; сумма $P(2)$ — блоками A_2, B_2, C_2. В свою очередь, сумма $P(A)$ будет сформирована информационными блоками A_0, A_1, A_2; сумма $P(B)$ — блоками B_0, B_1, B_2; сумма $P(C)$ — блоками C_0, C_1, C_2.

При такой организации можно преодолеть любые отказы двух дисков и многие отказы трех дисков. Однако при выполнении логической записи реально происходят **шесть обращений** к диску: за старыми данными, за контрольными суммами по строкам и по столбцам, а также для записи новых данных и новых значений контрольных сумм. Такая избыточность может оказаться приемлемой только для некоторых специализированных применений с очень высокими требованиями к отказоустойчивости.

Рис. 4.18. RAID уровня 6

В целом уровень RAID6 имеет очень высокую отказоустойчивость, большую скорость считывания, но из-за большого объема контрольной информации — низкую скорость записи. Кроме того, он сложен в реализации и характеризуется низким коэффициентом использования дискового пространства.

RAID7: отказоустойчивый массив с элементами оптимизации.

В этом массиве оптимизация используется для повышения производительности. RAID7 является запатентованным методом фирмы Storage Computer Corporation. Здесь используются концепции RAID3 и RAID4.

В состав RAID7 входит контроллер со встроенным микропроцессором под управлением ОС реального времени, позволяющей обрабатывать все запросы на передачу данных асинхронно и независимо. Блок вычисления контрольных сумм интегрирован с блоком буферизации. Для хранения информации о контрольных суммах используется отдельный диск (рис. 4.19).

Как следует из вышеизложенного, у каждого из уровней RAID имеются свои достоинства и недостатки. Поэтому инженерная мысль работает в направлении объединения достоинств различных видов массивов. С этой целью стали создавать так называемые **составные массивы**. Чаще всего для этого используется сочетание быстрого уровня RAID0 с достаточно надежными уровнями RAID1, 3 или RAID5. Получающийся составной массив обладает улучшенными характеристиками, но это увеличивает стоимость и усложняет реализацию.

Рис. 4.19. RAID уровня 7

В общем случае составной RAID-массив строится по следующей схеме: сначала диски делятся на наборы (SET), затем на основе каждого набора создаются простые массивы, которые потом объединяются в один мегамассив. Так, например, запись *X+ Y* для обозначения составного RAID-массива означает, что сначала диски объединяются в RAID-массив уровня *X*, а потом несколько массивов RAID X объединяются в RAID-массив уровня *Y*.

Наиболее распространенным в настоящее время являются составные массивы:

RAID01 (RAID0 + RAID1);
RAID10 (RAID1 + RAID0);
RAID03 (RAID0 + RAID3);
RAID30 (RAID3 + RAID0);
RAID05 (RAID0 + RAID5);
RAID50 (RAID5 + RAID0);

RAID15 (RAID1 + RAID5);
RAID51 (RAID5 + RAID1).

RAID01 часто называют «зеркалом страйков», а RAID10 — «страйком зеркал». К сожалению, этот американизм вытесняет нормальное русское слово «чередование». Основным недостатком составных массивов RAID01 и RAID10 является низкий процент использования емкости накопителей (примерно 50%).

Массивы RAID03 и RAID30 имеют довольно высокий процент использования емкости дисков и высокую скорость чтения данных, но достаточно дороги и сложны в реализации.

Массивы RAID15 и RAID51 позволяют значительно повысить надежность. Так, например, RAID15 продолжает работать при отказе трех накопителей, а RAID51 продолжает работать при потере 5 из 8 жестких дисков. За это приходится платить большим количеством неиспользуемой емкости дисков и общим удорожанием системы.

В ряде случаев возникает ситуация, когда нужно иметь один диск достаточно большой емкости и при этом не использовать ни зеркалирование, ни чередование, ни контрольные суммы. В этом случае применяют метод **JBOD (Just A Bunch of Disk)**, когда используется все дисковое пространство, и, кроме того, при выходе из строя одного из дисков информация на других дисках не повреждается.

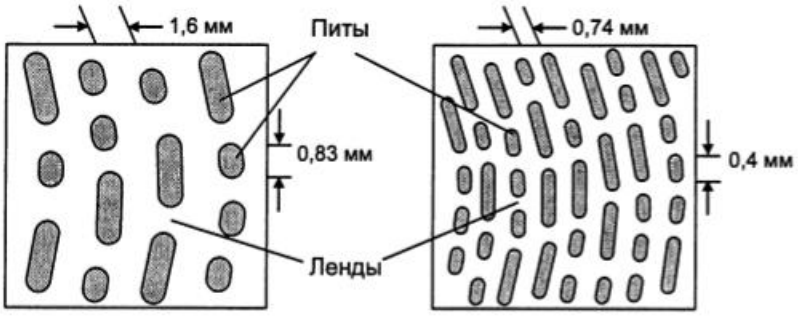
4.5.2.4. Оптические диски

Накопители на оптических дисках (НОД) являются относительно «молодым» видом ЗУ. Первые НОД появились в середине 80-х годов XX столетия. Хотя они появились из другой области (цифровая звукозапись), однако быстро заняли прочное место в иерархии внешней памяти ЭВМ.

В отличие от жестких дисков, где привод и носитель (диск) входят в состав одного устройства, НОД являются съемными. Диски являются самостоятельными компонентами, а привод представляет собой отдельное устройство.

Оптические диски изготавливаются из пластмассы, на поверхность которой наносится отражающий слой. Как для чтения, так и для записи НОД используются лазерные лучи, направляемые на поверхность вращающегося оптического диска. Луч лазера отражается от поверхности диска и фиксируется в фотоприемнике. При чтении информации используется различие интенсивности (или фазы) отраженного света, соответствующего нулю и единице данных. Различие интенсивности или фазы света возникает за счет **углублений** (ямок, питов — pits), формируемых в отражающем слое при записи, либо за счет изменения коэффициента отражения света от этого слоя (рис. 4.20). Промежутки слоя между питами называются **лендами**.

Питы и ленды образуют *единую спиральную дорожку*, расположенную (обычно) с одной стороны оптического диска и начинающуюся с его центральной части. Шаг дорожки и минимальный размер питов различается для разных НОД (см. рис. 4.20).



а) CD-ROM *б) DVD-ROM*

Рис. 4.20. Поверхность дисков CD и DVD

Следует заметить, что питы не являются битами 0 или 1 в привычном понимании. Каждый пит представляет сразу несколько бит информации. Наиболее часто используемая последовательность — это совокупность, в которой между двумя 1 не меньше двух и не больше десяти нулей. При этом питы чередуются в моменты, соответствующие началу бита, равного 1 (рис. 4.21).

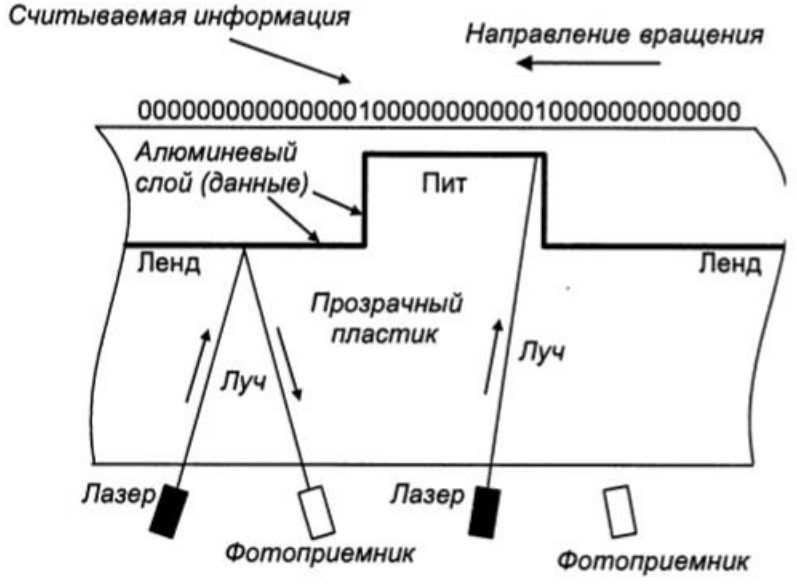


Рис. 4.21. Процесс чтения оптического диска

Питы и ленды с точки зрения представления информации равнозначны и характеризуют только временные интервалы между *двумя единицами* в последовательности бит. Поскольку состояние (ленд или пит) изменяется в моменты, соответствующие именно началу бита, равного 1, то протяженность каждого участка (и пита и ленда) лежит в пределах от 3 периодов тактовой частоты (так

называемый интервал 3T) до 11 периодов (интервал 11T), т. е. включает и длительность одного бита, равного **1**. От участка, который называется «ленд», свет лазера отражается и попадает в фотоприемник, а от участков «пит», свет из-за расфокусировки луча в фотоприемник не попадает. Фактически ленд — это вся поверхность диска, кроме питов. Для систем слежения за записью питы — это темные участки диска, а ленды — светлые. На рис. 4.21 схематично показано кодирование информации на оптическом диске, а также отражен процесс чтения диска.

Первые оптические диски получили название компакт-дисков или CD (Compact Disk), а поскольку они не допускали изменения записанной информации, то по аналогии с постоянными ЗУ (Read-Only Memory) их стали называть CD-ROM.

В настоящее время существуют несколько признаков, по которым можно классифицировать НОД. Наиболее часто используемыми являются:

- тип носителя;
- возможность изменения записанной информации;
- формат хранения данных;
- способ записи информации.

На рис. 4.22 приведен вариант классификации НОД по приведенным выше признакам.

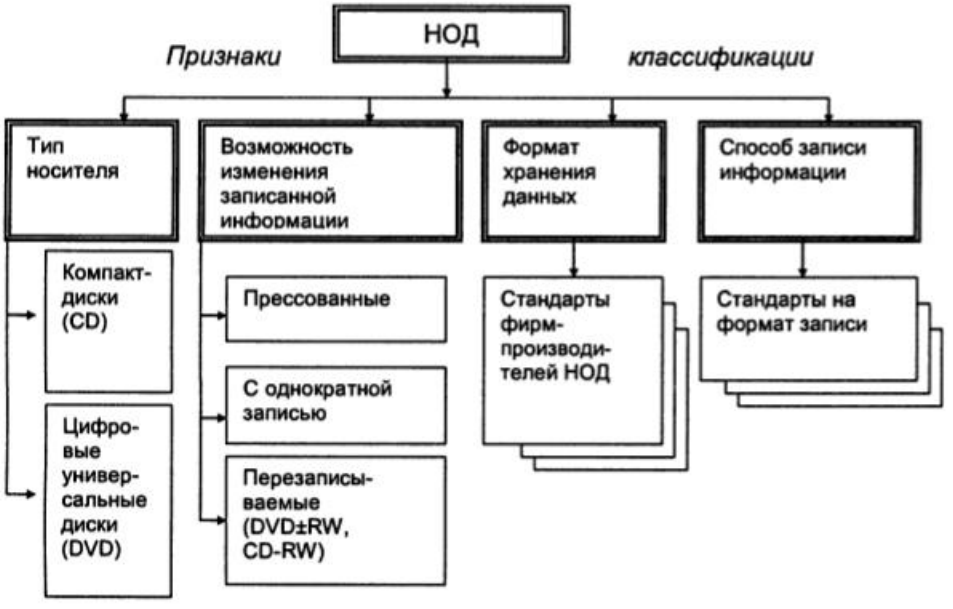


Рис. 4.22. Классификация НОД

По **типу носителя** различают компакт-диски (CD) и DVD-диски (Digital Versatile Disks — цифровые универсальные диски). DVD-диски можно рассматривать, как следующее поколение НОД, обеспечивающее более высокую плотность хранения информации и скорость передачи данных.

В стандартном варианте компакт-диски допускают запись до 700 Мбайт. DVD-диски внешне похожи на CD-диски и по размерам и по материалу, из которого изготовлена основа. Главное их отличие от CD-дисков заключается в организации хранения данных.

У DVD-дисков размеры питов и шаг спиральной дорожки примерно в два раза меньше, чем у CD-дисков. DVD-диски состоят из двух соединенных между собой половинок (сторон). DVD-диски могут быть *односторонними* и *двухсторонними*. Если на стороне диска имеется один слой для хранения информации, то такой диск называют *однослойным*, если имеются два слоя, то — *двухслойным*. Кроме того, способы кодирования информации у DVD- и CD-дисков различны. Емкость DVD-дисков может составлять до 17 Гбайт данных.

По возможности смены хранимой информации различают: **прессованные, с однократной записью** (записываемые — Recordable или R) и **перезаписываемые** (Rewritable или RW).

Прессованные CD диски имеют простой носитель, состоящий из трех слоев:

- пластмассовой (поликарбонатной) основы, на которой штампуется спиральная дорожка с ямками, несущая записанную информацию;
- отражающего слоя (алюминиевая металлизация);
- слоя лака (для защиты отражающего слоя от царапин и пыли). На этом слое наносится полиграфическое оформление диска.

Рабочая поверхность таких дисков, как правило, **белая**. Технология изготовления прессованных дисков напоминает технологию изготовления грампластинок, но гораздо сложнее и точнее.

Записываемые компакт-диски (CD-R) состоят из четырех слоев:

- пластмассовой (поликарбонатной) основы;
- записывающего слоя из специального красителя;
- отражающего металлизированного слоя (золото или серебро);
- защитного слоя лака, на котором наносится полиграфическое оформление.

Записываемые компакт-диски имеют цвета зеленоватых или синих оттенков. Похожую структуру имеют записываемые диски DVD, но при этом данные диски могут иметь два слоя (рис. 4.23).

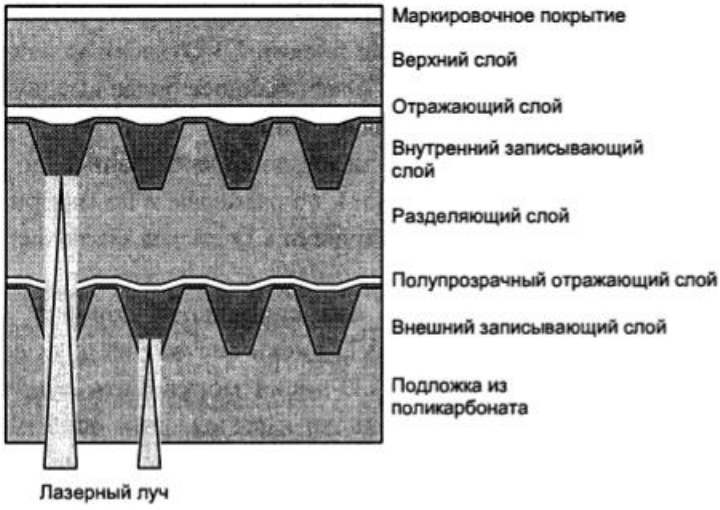


Рис. 4.23. Структура записываемого двухслойного диска DVD-R (DL)

Перезаписываемые диски CD-RW или DVD-RW имеют еще более сложную структуру. Чаще всего они состоят из шести слоев:

- пластмассовой (поликарбонатной) основы;
- слоя диэлектрического материала (смесь серебра, иридия, теллура и антимония);
- отражающего слоя металлизации (золото, серебро);
- еще одного слоя диэлектрического материала;
- защитного слоя лака.

Диэлектрические слои служат для отвода тепла при записи информации.

Очень часто в качестве активного слоя используют красители **цианина** и **фталоцианина** или их гибридной смеси.

CD-RW диски, как правило, имеют сероватый цвет поверхности, хотя возможны цвета различных оттенков.

Прессованные DVD-диски обозначают как DVD-5, DVD-9, DVD-10, DVD-18, что примерно соответствует их информационной емкости в гигабайтах. Однократно записываемых и перезаписываемых вариантов DVD-дисков несколько, что является следствием конкурентной борьбы производителей, каждый из которых продвигал на рынок свой стандарт. В настоящее время устоялись два формата — так называемые «**плюс**» (DVD+R, DVD+RW) и «**минус**» (DVD-R, DVD-RW). Большинство современных приводов DVD способны читать и записывать оба этих формата.

Данные записываются и хранятся на НОД в различных форматах в зависимости от области их использования. На каждый формат разработан, как правило, свой стандарт. Особенно много таких стандартов для компакт-дисков.

Первый стандарт на компакт-диски был разработан в 1980 г. компаниями Philips и Sony и описывал диски, предназначенные для цифровой записи музыки.

Он получил название **CD-DA** (Compact Disk Digital Audio). В стандарте используется формат оцифровки звука: 16-битное аналого-цифровое преобразование с частотой снятия отсчетов 44,1 КГц.

Данные записываются на диски секторами, которые являются минимально адресуемыми единицами информации. Сектор (блок) имеет емкость 2352 байта и состоит из 96 кадров по 24 байта в каждом. У CD-ROM только 2048 или 2336 байтов являются информационными, остальные — служебные. Число служебных байтов зависит от способа записи. Фактически емкость блока больше, так как для уменьшения ошибок чтения/записи используются различные корректирующие коды. Поэтому реальная емкость блока в 3–3,5 раза больше указанной выше. Логически секторы объединяются в дорожку (трек), содержащую до 300 секторов.

НОД считаются долговечными, но и они подвержены разрушению. Несмотря на то что они устойчивы к воздействию магнитного поля, могут без ущерба для информации содержать неглубокие царапины и мелкие повреждения, следует помнить, что все НОД имеют пластмассовую аморфную основу. Поэтому НОД чувствительны к высокой температуре, а со временем отражающий слой, как и любой аморфный материал, может «потечь», что вызовет потерю информации.

4.5.2.5 Флэш-память

Флэш-память является особым видом энергонезависимой, перезаписываемой полупроводниковой памяти. Впервые флэш-память была разработана компанией Toshiba в 1984 г. В 1989–1990 гг. компания Toshiba для названия нового вида памяти употребила слово «flash» в контексте «быстрый, мгновенный» при описании своих микросхем.

Сфера применения флэш-памяти весьма обширна. Она используется в качестве носителя микропрограмм, для хранения BIOS в ПК, в КПК, принтерах, видеоплатах, мобильных телефонах и т.п. Одним из преимуществ флэш-памяти является отсутствие движущихся механических частей, что значительно повышает ее надежность.

Исторически флэш-память происходит от ROM (Read Only Memory) памяти и функционирует подобно RAM (Random Access Memory) памяти.

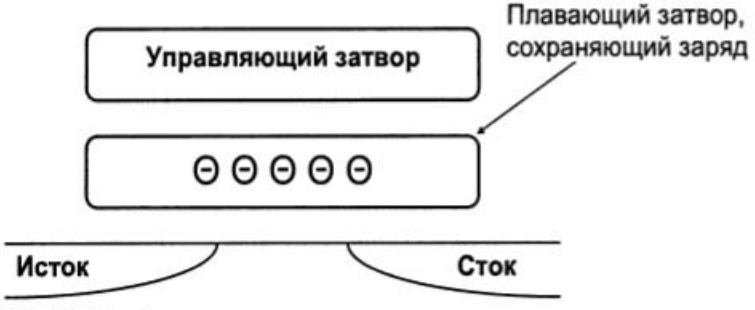


Рис. 4.24. Ячейка флэш-памяти на одном транзисторе

Данные во флэш-памяти хранятся также в ячейках, похожих на ячейки RAM-памяти, но в отличие от последней при отключении питания информация не пропадает. Элемент, хранящий информацию, показан на рис. 4.24. Это один транзистор, у которого под управляющим затвором помещен еще так называемый *плавающий затвор* (из электрически изолированного поликремния), позволяющий хранить заряд в виде электронов. Количество заряда определяет работу этого транзистора. Это различие в поведении определяет состояние ячейки. Наличие заряда на транзисторе понимается как логический 0, а его отсутствие — как логическая 1.

При чтении логической 1 (рис. 4.25) в отсутствие заряда на плавающем затворе образуется *n*-канал в подложке между истоком и стоком, и возникает ток.

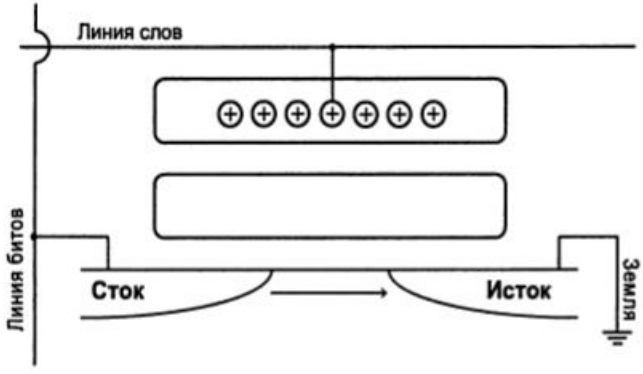


Рис. 4.25. Чтение логической 1 из ячейки флэш-памяти

Наличие заряда на плавающем затворе (рис. 4.26) меняет вольт-амперные характеристики транзистора таким образом, что при обычном для чтения напряжении канал не появляется, и тока между истоком и стоком не возникает.

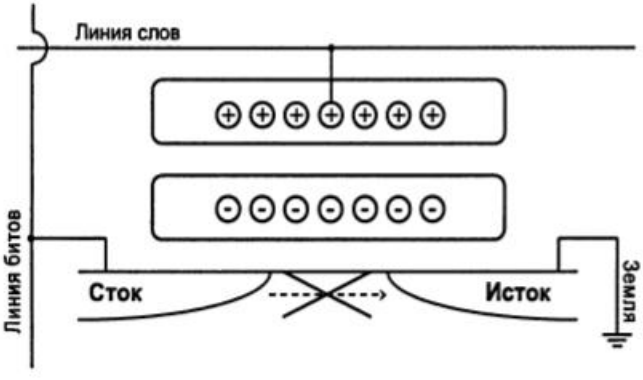


Рис. 4.26. Чтение логического 0 из ячейки флэш-памяти

При программировании (рис. 4.27) на сток и управляющий затвор подается высокое напряжение (причем на управляющий затвор напряжение подается при-

близительно в два раза выше). «Горячие» электроны из канала инжектируются на плавающий затвор и изменяют вольт-амперные характеристики транзистора. Такие электроны называют «горячими» потому, что они обладают высокой энергией, достаточной для преодоления потенциального барьера, создаваемого тонкой пленкой диэлектрика.

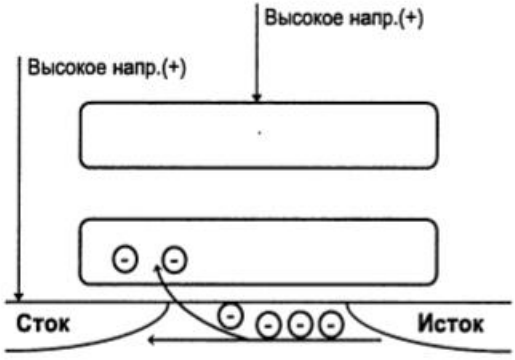


Рис. 4.27. Программирование ячейки флэш-памяти

При стирании (рис. 4.28) высокое напряжение подается на исток. На управляющий затвор подается высокое отрицательное напряжение. Электроны туннелируют на исток. **Эффект туннелирования** — один из эффектов, использующих волновые свойства электрона. Сам эффект заключается в преодолении электроном потенциального барьера малой «толщины». Для наглядности представим себе структуру, состоящую из двух проводящих областей, разделенных тонким слоем диэлектрика (обедненная область). Преодолеть этот слой обычным способом электрон не может — не хватает энергии. Но при создании определенных условий (соответствующее напряжение и т. п.) электрон проскакивает слой диэлектрика (туннелирует сквозь него), создавая ток. Важно отметить, что при туннелировании электрон оказывается «по другую сторону», *не проходя через диэлектрик*.

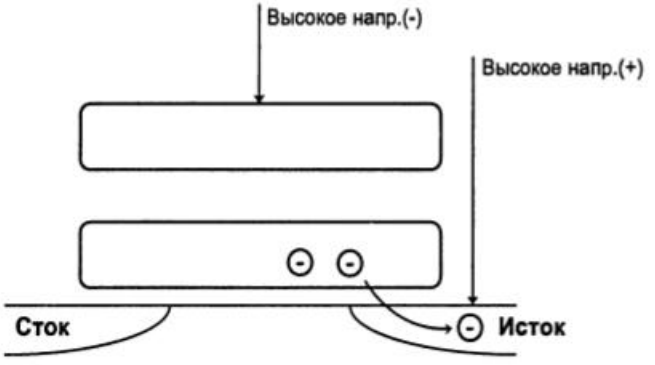


Рис. 4.28. Стирание ячейки флэш-памяти

Обратим внимание на то, что поведение транзистора флэш-памяти зависит от количества электронов. Если заряд плавающего затвора у однобитного транзистора меньше 5000 электронов, то это означает, что ячейка хранит логическую **1**, а если заряд больше 30000 электронов, то — **0**. Заряд ячейки вызывает изменение порогового напряжения транзистора, и при операции чтения измеряется величина этого порогового напряжения, а по нему определяется количество заряда на плавающем затворе.

Возможность сохранять заряд на ячейке дает возможность сохранять несколько бит на одной ячейке. Флэш-ячейка является аналоговым запоминающим устройством, а не цифровым. Она хранит заряд, а не биты. Поэтому, используя контролируемый метод программирования, на плавающий затвор можно поместить точное количество заряда. При установке заряда в одно из четырех состояний можно запрограммировать два бита данных на одной ячейке. Каждое из четырех состояний соответствует одному из двухбитных наборов. Данная технология получила название MLC (Multilevel Cell — многоуровневая ячейка).

Заметим, что потребление энергии во время работы флэш-памяти примерно в 10–20 раз меньше, чем других носителей информации с движущимися механическими частями.

Несмотря на очевидные преимущества, флэш-память пока не способна вытеснить ни динамическую оперативную память, ни дисковые накопители. Во-первых, флэш-память работает медленнее и, во-вторых, она имеет ограничение по количеству циклов перезаписи (от 10 000 до 1 000 000 для разных типов). Процедуры записи/стирания информации приводят к изнашиванию ячеек флэш-памяти. Поэтому в микросхемах памяти используются для этих целей специальные алгоритмы, обеспечивающие равномерное использование всех ячеек памяти в процессе ее функционирования.

Очень широкое распространение получили так называемые **флэш-диски** (flash drive), которые подключаются к ЭВМ через шину USB. При соответствующей поддержке операционной системы организованная таким образом флэш-память представляется как съемный дисковый накопитель. При этом объемы флэш-памяти, выпускаемой разными фирмами, различны и составляют от сотен мегабайт до нескольких гигабайт.

4.5.2.6. Устройства резервного копирования информации

Средства резервного копирования предназначены для создания копий данных, необходимых для восстановления работоспособности ЭВМ или ВС в случае аварийной ситуации или других непредвиденных обстоятельств. При наличии резервной копии можно достаточно быстро восстановить данные на момент последнего резервного копирования (т. е. после восстановления теряются все изменения, которые произошли в системе с момента последнего резервного копирования).

Обычно резервное копирование касается всех данных, находящихся на жестких дисках, в том числе различной системной и служебной информации. Сами же

устройства и системы резервного копирования находятся на более низком уровне иерархии памяти ЭВМ (см. рис. 3.1). Такое структурированное построение предназначено для снижения стоимости хранения редко используемой информации, так как на носители более низких уровней обычно автоматически перемещаются файлы, к которым пользователи не обращались в течение длительного времени. При этом перемещение файлов организуется таким образом, чтобы объем свободного пространства на жестких дисках ЭВМ или ВС поддерживался на заданном уровне.

Существуют несколько основных видов устройств и систем резервного копирования, предназначенных для применения как на малых предприятиях, так и в крупных корпорациях с распределенными филиалами. Все они отличаются друг от друга по ряду характеристик, и не в последнюю очередь по степени автоматизации операций резервного копирования.

На протяжении почти полувека наиболее распространенными устройствами резервного копирования остаются *накопители на магнитной ленте* (НМЛ), которые постоянно видоизменяются по мере развития технологий. В настоящее время существуют следующие типы НМЛ и систем резервного копирования:

Ленточные накопители — стримеры (streamer от англ. stream — «поток»), которые записывают (и считывают) информацию на картридж или кассету.

Стекеры (stacker от англ. stack — «скирдовать, собирать в кучу»), в которые установливается один стример и несколько картриджей. При этом картриджи устанавливаются в стекер в специальных лотках и подаются в стример в жестко установленном порядке при помощи специального роботизированного механизма.

Автозагрузчики (autoloader), в определенной степени напоминающие стекеры. Отличительной чертой автозагрузчиков является возможность подачи картриджей (размещаемых в специальном магазине) в произвольном, а не в жестком порядке.

Ленточные библиотеки (libraries) — это еще более сложные решения для резервного копирования на магнитную ленту, представляющие собой системы, в состав которых входят много картриджей и до нескольких стримеров. Библиотека состоит из **отсеков**, в которых хранятся картриджи, и **механизма смены** (jukebox) картриджей в стримерах. Соответственно существуют две схемы загрузки картриджей роботизированным механизмом. При использовании первой схемы любой картридж загружается в любой стример. Во второй схеме каждый стример работает только с конкретными картриджами. Наиболее важными характеристиками ленточных библиотек являются *скорость передачи данных*, *емкость* и *надежность*. Ленточные библиотеки отличаются друг от друга такими характеристиками, как число слотов для картриджей и число стримеров, количество портов доступа к картриджам, перечень поддерживаемых операционных систем, а также возможностями встроенных функций управления. Библиотеки — это очень мощ-

ные решения для резервного копирования, позволяющие с большой скоростью осуществлять резервное копирование огромных объемов данных (достигающих сотен терабайт).

Массивы RAIT (Redundant Arrays of Independent Tape) — это так называемые массивы стримеров с избыточностью (их еще называют избыточными массивами независимых стримеров). В корпусе RAIT-массива находятся несколько стримеров, каждый из которых одновременно работает только с одним картриджем, за счет чего существенно увеличиваются скорость резервного копирования и отказоустойчивость. RAIT-массивы созданы на базе спецификаций RAID, применяемых для дисковых подсистем. Платой за высокую скорость и надежность является малая емкость RAIT-массивов и невозможность автоматической смены (ротации) носителей.

Помимо ленточных, существуют и другие типы устройств и систем резервного копирования, работающие с носителями информации других типов (в первую очередь магнитооптическими и оптическими). Достаточно широко распространены, например, магнитооптические и оптические библиотеки, достоинством которых является огромный жизненный цикл их картриджей (несколько миллионов записей/перезаписей) и длительный срок эксплуатации (более 30 лет). Недостатком магнитооптических библиотек является относительно небольшая емкость носителей. Существуют также библиотеки на базе CD- и DVD-дисков.

В качестве носителей магнитных лент выступают **кассеты** и **картриджи**. Кассета представляет собой корпус с двумя бобинами, на которые намотана магнитная лента, и напоминает кассету бытового видеомагнитофона. Картридж имеет только одну бобину, на которую намотана магнитная лента. Поэтому в продольном сечении он имеет форму квадрата. Приемная бобина находится внутри накопителя (стримера). В момент загрузки картриджа в стример свободный конец магнитной ленты захватывается специальным устройством и наматывается на приемную бобину. Типичная емкость современных носителей составляет сотни гигабайт при скорости передачи данных в несколько десятков мегабайт в секунду.

Запись на магнитную ленту может производиться двумя методами: *наклонным* и *линейным серпантинным*. В системах **наклонной записи** несколько считывающих/записывающих головок размещают на вращающемся барабане, установленном под углом к вертикальной оси (аналогичная схема записи применяется в бытовой видеоаппаратуре). Движение ленты при записи/чтении может осуществляться только в одном направлении. Лента медленно протягивается, что при быстро вращающемся барабане позволяет сочетать высокие плотность и скорость записи с повышенной надежностью.

В системах **линейной серпантинной записи** считывающая/записывающая головка неподвижна при движении ленты. Данные на ленте записываются в виде множества параллельных дорожек (серпантина). Головка размещается на специальной подставке, при достижении конца ленты она сдвигается на другую дорожку. Движе-

ние ленты при записи/чтении осуществляется в обоих направлениях. Обычно устанавливаются несколько головок, образующих несколько каналов записи/чтения.

Кратко рассмотрим технологии чтения/записи на магнитную ленту, получившие широкое распространение в системах резервного копирования и архивирования.

Первые реализации технологии *DDS* (*Digital Data Storage*) были основаны на использовании цифровых аудиокассет (*Digital Audio Tape, DAT*), применяемых в цифровых магнитофонах, поэтому иногда данную технологию продолжают называть DAT. Стандарт на технологию DDS был разработан в 1989 г. компаниями HP и Sony.

В качестве носителя в технологии DDS используется кассета с магнитной лентой шириной ~ 4 мм (0,15″). Метод записи — наклонный. Для DDS характерны очень небольшие размеры и вес: достаточно сказать, что кассета чуть больше спичечной коробки. Кроме этого, к достоинствам технологии следует отнести низкие цены на накопители и кассеты. Недостатками являются сравнительно небольшая емкость кассеты и невысокая скорость передачи данных.

В основе технологии *DLT* (*Digital Linear Tape*) лежит многоканальный метод работы с магнитными лентами — высокая скорость работы с лентами достигается за счет одновременного чтения или записи нескольких головок устройства. Запись производится линейным серпантинным методом. В DLT-стримерах применяется картридж с магнитной лентой шириной полдюйма, причем лента заполняет весь объем картриджа. Технология DLT сейчас наиболее распространена в системах резервного копирования среднего уровня и выше.

Примером DLT-устройств могут служить НМЛ семейства Mammoth фирмы Exabyte. Существует также формат SuperDLT (SDLT). В устройствах SuperDLT применяются другая, более совершенная магнитная лента, иные магнитные головки, оптическая система позиционирования дорожек и др. Устройства SDLT не могут работать с картриджами DLT из-за своих конструктивных особенностей.

В конце 90-х годов прошлого века был разработан перспективный формат записи на магнитную ленту — *LTO* (*Linear Tape Open*). Он был создан совместными усилиями компаний IBM, HP и Seagate. Следует отметить, что формат LTO разрабатывался, не базируясь на каком-либо уже существующем решении для резервного копирования. Первые стримеры формата LTO были выпущены в декабре 2000 г., а начиная с января 2003 г., на рынке появились модели стримеров LTO уже второго поколения. В основе LTO лежит двунаправленная многоканальная технология линейной записи, в которой используется магнитная лента шириной полдюйма.

Отметим также 8-миллиметровую технологию *AIT* фирмы Sony (характерной чертой кассет AIT является наличие в них встроенной памяти) и технологию *QIC* фирмы Tandberg, основанную на использовании четвертьдюймовых лент с линейной записью.

4.5.3. Устройства ввода информации

4.5.3.1. Клавиатура

Для ручного ввода символьной информации в основном используется клавиатура. **Клавиатура** — устройство ручного ввода данных в ЭВМ, кодируемых путем нажатия расположенных на ней клавиш. К конструкциям клавиатур предъявляется ряд общих технических требований: эргономических, функциональных, эксплуатационных, надежностных, стоимостных и др. Основные требования к техническим параметрам клавиатур ввода приведены в табл. 4.2.

Клавиатуры обычно реализуются в виде функционально и конструктивно законченного узла — электронного пульта, размещенного в отдельном плоском корпусе. Клавиатура, как правило, может свободно перемещаться на столе пользователя, соединяться с системным блоком гибким кабелем, шнуром или с использованием методов беспроводной связи. Для удобства пользователя наклон плоских клавиатур на столе может регулироваться в пределах нескольких градусов. Современные клавиатуры позволяют работать с несколькими фонетическими системами (алфавитами и буквами). Состояние клавиатуры (например, указание о регистре вводимых символов) осуществляется с помощью световой индикации (обычно светодиодной).

Таблица 4.2

Основные требования к техническим параметрам клавиатур ввода данных

Параметр	*Значение*
Число клавиш с кодовым выходом, не более	128
Максимальная частота нажатия клавиш в 1 сек, не менее	10
Средняя наработка на отказ нажатий, не менее	10^8
Средний срок службы, лет	10
Усилие нажатия клавиш, Н:	
Одиночных	0,35...1,00
Широких	0,35...2,00

Клавишные переключатели, преобразующие механическое движение руки человека в электрический сигнал, — одни из основных элементов конструкции клавиатуры. Они, как и элементы электронных схем, размещаются на поверхности печатной либо другой несущей платы. Современные клавиатуры содержат несколько десятков клавиш с нанесенными на них различными символами (буквами, цифрами, специальными знаками).

Для упрощения работы с клавиатурой используется принцип кодирования клавиш, не зависящий от кодирования символов. В системное устройство ЭВМ посылается не код символа, которому соответствует данная клавиша, а код самой клавиши (позиционный код). Интерпретация сигнала нажатой клавиши осущест-

вляется программным путем, что обеспечивает гибкость и мобильность при использовании различных фонетических и даже семиотических систем. Переход к коду символа осуществляется специальной системой управления клавиатурой, поэтому можно с помощью специальных прикладных программ перепрограммировать те или иные клавиши для других целей.

Технические и эргономические характеристики клавиатур ЭВМ, удобство пользования, максимальная скорость набора, показатели надежности (например, количество переключений) и ряд других — во многом определяются техническими характеристиками клавишных переключателей. Важны форма и размер клавиш, шаг их расположения, величина хода, усилия нажатия, окраска, надписи и прочие характеристики.

В зависимости от требований, предъявляемых к клавиатурам, в них используются клавишные переключатели различных типов. По способу управления клавишные переключатели делятся на клавиши нажатия (тактильные) и прикосновения (сенсорные). Применяются как *контактные* (с электрическим контактом), так и *бесконтактные* переключатели.

Контактные переключатели, в которых электрическое замыкание и размыкание цепи электромеханического контакта происходит после нажатия клавиши, просты в конструкции, обеспечивают низкое омическое сопротивление контакта, дешевы. Однако такие переключатели не могут обеспечить в цепи четкого фронта сигнала замыкания (размыкания). В момент нажатия клавиши, стоящей в электрической цепи, происходит нерегулярное замыкание этой цепи («дребезг» контакта) до тех пор, пока не установится надежный сигнал. Влияние «дребезга» устраняют схемным либо программным путем. К недостаткам контактных переключателей можно отнести сравнительно малый срок службы (число переключений до отказа), непостоянство переходного сопротивления контакта вследствие окисления покрытий контактирующих поверхностей под воздействием окружающей среды или электроэрозии. Лучшие показатели по долговечности дают герконовые переключатели. В них коммутация герметизированных контактов осуществляется под воздействием миниатюрного постоянного магнита, приводимого в движение клавишным стержнем.

При изготовлении клавиатур ЭВМ широко используются контактные переключатели на основе токопроводящей резины, в которых ряд элементов клавиши изготавливается из силиконовой резины. На нижней поверхности колпачка клавиши из непроводящей резины помещается прокладка из токопроводящей резины. Клавиши закрепляются на плате, на которой печатным способом выполнены необходимые электрические контакты и схемы клавиш. В момент нажатия клавиши токопроводящая резиновая прокладка прижимается к плате и замыкает необходимые контакты. Конструкции переключателей на основе токопроводящей резины обеспечивают, как правило, малый ход клавиш.

Мембранные клавиатуры отличаются хорошей герметичностью, малыми размерами и массой, помехозащищенностью, малым «дребезгом», незначительным расходом драгоценных металлов. Они просты по конструкции, содержат минимальное число деталей, обладают высокой степенью автоматизации и возможностью крупносерийного производства. Мембранная клавиатура состоит из соединенных в пакет пластмассового корпуса и мембранной панели, состоящей из двух гибких лавсановых контактных пластин (печатных плат) с металлизированными токопроводящими головками, являющимися контактирующими и токоведущими частями переключателя. Гибкие печатные платы изолированы друг от друга диэлектрической пластиной и находятся на металлическом основании. При нажатии на клавишу усилие через демпфирующую пружину, расположенную в корпусе, передается на гибкие печатные платы, замыкая их контакты между собой. С помощью пружины клавиша возвращается в исходное положение.

В случае, когда требуется обеспечить высокую надежность работы, бесшумную эксплуатацию, а также устойчивость против влаги, коррозии, пыли, а стоимость не играет определяющей роли, в конструкциях клавиатур применяются бесконтактные клавишные переключатели.

В бесконтактных переключателях механическое воздействие (перемещение или прикосновение) вызывает изменение емкости, индуктивности или сопротивления, а затем это изменение преобразуется в электрический сигнал. К бесконтактным относятся также сенсорные и оптоэлектронные переключатели.

Рассмотрим клавишные переключатели, использующие изменение емкости при нажатии. На печатную плату в виде прямоугольных пластинок наносятся обкладки конденсатора, сверху на обкладки наносится слой изолятора, например лавсановая пленка. На одну из обкладок от генератора подается специальный сигнал с определенной частотой и амплитудой. При нажатии на клавишу соединенная с ней металлическая пластина опускается на обкладки конденсатора, в результате его емкостное сопротивление уменьшается, и на второй обкладке появляется сигнал, поступающий на электронную схему.

Клавишный переключатель индуктивного типа состоит из неподвижной (направляющей) и подвижной (штока) частей. На одном конце штока находится подвижная часть датчика — металлическая пластина, которая прижимается к штоку за счет пружины. На другом конце на шток насажена клавиша. Неподвижная часть датчика электрически представляет собой индуктивность и выполнена печатным способом. Для предотвращения механического износа эта часть защищается склеивающей лентой. При нажатии на клавишу металлическая пластина приближается к индуктивности, вызывая уменьшение значения последней. При изменении индуктивности формируется сигнал о срабатывании датчика. Клавиша возвращается в исходное состояние за счет пружины, прижимающей металлическую пластину к штоку.

В ряде клавиатур используются сенсорные переключатели, которые не имеют подвижных механических частей (отсутствует ход клавиши). Прикосновение руки пользователя к клавише сразу вызывает переключение. Работа с такой клавиатурой требует от пользователя определенных навыков и звуковой и световой обратной связи.

4.5.3.2. Манипуляторы «мышь», джойстик, световое перо

Манипулятор «мышь» является одним из простейших, но удобных средств ввода графической информации. С его помощью легко вводить данные, графику, «раскрашивать» изображение, перемещать курсор или элемент изображения по экрану дисплея и т. п. Манипулятор представляет собой настольный прибор, предназначенный для преобразования его перемещения в двух ортогональных направлениях (*X* и *Y*) в серии электрических импульсов. «Мышь» может подключаться к ЭВМ как с помощью тонкого гибкого кабеля, так и используя беспроводную технологию (например, с помощью инфракрасного порта). Используются в основном два принципа работы устройства: механический (оптомеханический) и оптический.

Основой конструкции механического манипулятора типа «мышь» является *преобразователь (основание с шаром)*, который вместе с платами с ИМС и *управляющими кнопками (клавишами)* помещается в пластмассовый корпус.

Адаптер манипулятора «мышь» принимает от манипулятора импульсные сигналы, сопровождающие перемещение манипулятора в направлениях *X* и *Y*, и преобразует их в цифровой код. С помощью адаптера определяется направление и перемещение манипулятора по рабочей поверхности и выдается информация устройству, управляющему курсором на экране видеомонитора. Пересчет текущих позиций манипулятора осуществляется центральным процессором. Приемопередатчик, реализуемый, как правило, на трехстабильных двунаправленных элементах, обеспечивает сопряжение шины данных, адаптера и системной шины.

Оптический манипулятор «мышь» перемещается по поверхности с произвольным рисунком, практически по любой неоднотонной поверхности. Источник излучения направляет луч света на поверхность, и отраженный луч попадает на фотоприемник, связанный с микропроцессором. Перемещение манипулятора, так как поверхность неоднотонна, вызывает соответствующие изменения в отражаемом луче. Микропроцессор определяет относительное перемещение манипулятора и соответствующее положение курсора на экране дисплея и передает информацию в ЭВМ.

Преимущество такой «мыши» — достоверность и надежность. Уменьшение количества механических узлов приводит к увеличению ее срока службы. Недостаток заключается в том, что при повреждении или износе покрытия «мышь» производит на мониторе хаотичное перемещение курсора.

Как уже указывалось, «мышь» может быть соединена с ЭВМ посредством инфракрасного порта, наподобие пультов дистанционного управления телевизором. Рядом или на компьютере устанавливается приемник инфракрасного излучения,

который кабелем соединяется с ЭВМ. Движение «мыши» преобразуется в инфракрасный сигнал, который затем передается в приемник. Для безупречной передачи инфракрасного сигнала всегда должен быть установлен «зрительный» контакт между приемником и передатчиком. Нельзя загораживать излучатель такой «мыши» книгами, теплопоглощающими или другими материалами. Альтернативой является передача информации от «мыши» посредством радиосигнала.

Устройством ввода, которое заняло прочную позицию прежде всего в области компьютерных игр, является **джойстик**. Различают цифровые и аналоговые джойстики. Цифровой джойстик оснащается 9-контактным Sub-D-разъемом (гнездо), аналоговый джойстик — 15-контактным двухрядным разъемом (вилка), который подключается к игровому порту. Использование цифрового джойстика требует установки в компьютер специальной карты или применения переходника с 9-контактного на 15-контактный разъем.

Джойстик представляет собой управляющую ручку с одной или несколькими кнопками. Цифровой джойстик реагирует на положение управляющей ручки (влево, вправо, вверх, вниз) и статус кнопки. Аналоговые джойстики регистрируют минимальные движения ручки управления, что, разумеется, обеспечивает более точное управление игрой. Точность управления можно увеличить, используя обработку таких сообщений, как поворот ручки управления на пол-оборота направо и налево, наискосок вниз или вверх. Подобная точность управления крайне важна для игр, в которых подвижные объекты должны точно позиционироваться.

Световые перья применяются довольно редко, поэтому ограничимся рассмотрением способа их действия и области применения. Световое перо похоже на шариковую ручку, в которую вместо пишущего шарика вмонтирован фотоэлемент. Кроме того, в стержне находится электронная составная часть, которая оценивает сигналы. В зависимости от исполнения световое перо оснащается одной или более кнопками, которые выполняют функции, схожие с функциями кнопок «мыши». Световое перо функционирует только совместно с монитором. При прикосновении стержнем к поверхности экрана электронное излучение регистрируется фотосенсором светового пера. Так как экран монитора состоит из множества **точек («пикселов»)**, то полученный сигнал можно передать на графическую карту, которая вычислит координаты электронного луча по времени его регистрации.

Теоретически световое перо может заменить «мышь», но на практике, если для управления крупными объектами световое перо еще надежно в применении, то при выборе мелких объектов оно не удобно.

Другая область применения светового пера — его совместное использование с **дигитайзером**. В этом случае световое перо выполняет «пишущую» функцию. Этот способ ввода информации поддерживается различным программным обеспечением.

Для профессиональных графических работ дигитайзер (со световым пером) практически является стандартным устройством, так как он с помощью соответ-

ствующих программ позволяет преобразовывать передвижение руки оператора в формат векторной графики.

Первоначально дигитайзер был разработан только для приложений САПР, потому что в этом случае необходимо определять и задавать точное значение координат большого количества точек. Это функциональное требование при использовании обычных устройств ввода (таких, как клавиатура) затруднительно, а при использовании «мыши» может быть выполнено неточно.

В то время как «мышь» может интерпретировать только относительные координаты, дигитайзер способен точно определять и обрабатывать абсолютные координаты. Для этого используется специальный планшет, который, помимо того что является рабочей («письменной») поверхностью, имеет еще и другие многочисленные функции, позволяющие непосредственно управлять соответствующими программами. Собственно в качестве средства ввода информации служат или световое перо или (чаще) круговой курсор, с помощью которого выполняется позиционирование и можно очень точно определять координаты на планшете.

4.5.3.3. Сканеры

Широкую популярность приобрели сканирующие устройства ввода — **сканеры** (от англ. scan — внимательно просматривать). В основе всех сканеров лежит общий принцип — преобразование двумерного изображения в электрические сигналы.

Первое устройство сканирующего типа было запатентовано в 1843 г. А. Бэйном (A. Bain). Маятник, закрепленный так, что он мог колебаться только в одной плоскости, подвешивался над электромагнитом. На маятнике закреплялся токосъемник, а контур сканируемого изображения наносился на лист бумаги токопроводящим веществом. При каждом касании контура токосъемником в подключенном к нему проводе начинал течь ток. Частота колебаний маятника задавалась расположенным под ним электромагнитом, при каждом колебании токосъемник считывал «дугообразную» линию, одновременно после каждого прохода токосъемника оригинал смещался по вертикали.

На принимающей стороне использовался точно такой же аппарат, работавший синхронно с передающим: в нем был закреплен лист бумаги, покрытый веществом, темнеющим под воздействием электрического тока. В первый год эксплуатации система Бэйна передала из Парижа в Лион свыше 5000 изображений.

Принцип работы современных сканеров следующий. В большинстве случаев источник света (светодиоды или лазер) освещает рабочее поле вводимого документа. Чувствительный к свету рабочий орган движется по изображению (или изображение движется относительно неподвижного рабочего органа) и за счет отраженного света считывает элементы изображения, обычно построчно, которые затем распознаются ЭВМ и преобразуются в файл, размещаемый в памяти машины. В рабочем органе используются датчики, преобразовывающие оптическое изображение в электрический сигнал с разрешающей способностью 10 ...30 точек/мм и выше. При преобразовании полутоновых графических изображений фиксируется града-

ции яркости. Для различения трех основных цветов (красного, зеленого и синего) применяются цветовые фильтры и источники с различным спектром излучения.

Сканеры позволяют вводить в компьютер в графическом виде текст, рисунки, слайды, фотографии и др. Технология считывания данных реализуется на основе использования светочувствительных датчиков двух типов: **приборов с зарядовой связью (ПЗС)** или **фотоэлектронных умножителей (ФЭУ)**.

ПЗС — это твердотельный электронный компонент, состоящий из множеств миниатюрных датчиков, которые преобразуют интенсивность падающего на них света в пропорциональный ей электрический заряд. В основу ПЗС положена чувствительность проводимости $p-n$-перехода обыкновенного полупроводникового диода к степени его освещенности. На $p-n$-переходе создается заряд, который уменьшается со скоростью, зависящей от освещенности. Чем меньше заряд, тем больше ток через диод.

Основанные на ламповой технологии **ФЭУ** осуществляют электронное усиление интенсивности отраженного от оригинала света. Попадая на катод ФЭУ, свет выбивает из него электроны, которые, проходя через пластины диодов, вызывают вторичную электронную эмиссию. Коэффициент усиления зависит от свойств материала и количества диодов. Напряжение, пропорциональное освещенности катода ФЭУ, снимается с анода и затем преобразуется в цифровой код.

Неотъемлемой частью любого сканера являются **аналого-цифровые преобразователи (АЦП)**. Они предназначены для преобразования непрерывно изменяющихся значений напряжения, получаемых с помощью ПЗС или ФЭУ, в числа, соответствующие оттенкам цвета или градациям серого. Качество сканированного изображения напрямую связано с разрядностью используемого в сканере АЦП. В современных сканерах разрядность АЦП составляет от 8 до 12 бит.

Наиболее важная характеристика сканера— **максимальное разрешение**, измеряемое предельным числом линий (точек) на единицу длины, которые могут быть распознаны оптической системой сканера. Однако следует учитывать, что изготовители сканеров часто подменяют понятия, называя разрешением иную характеристику— **шаг дискретизации**.

Так как число и плотность размещения элементов ПЗС-датчика фиксировано, именно оно определяет максимальное количество точек изображения на единицу площади оригинала. Современные сканеры содержат ПЗС-датчики с 300, 600, 1200 и 2400 элементами на дюйм, что позволяет производителям говорить о «разрешении» 300, 600, 1200 и 2400 точек/дюйм.

Однако на реальное разрешение сканера оказывают влияние многие другие факторы: качество оптической системы и подсветки, уровень шума ПЗС-датчика, взаимные наводки его элементов, точность механизма позиционирования и др. Поэтому на практике *реальное разрешение заметно меньше плотности размещения элементов ПЗС-датчика*. Реальное разрешение может быть установлено только путем экспериментов, для которых необходимы специальные эталонные оригиналы.

ПЗС позволяет регистрировать только уровень интенсивности света, а не его частоту, т. е. получаемое им изображение монохромно. Для получения цветного изображения применяются множество **методов цветоделения**, которые можно разделить на три группы: *с применением фильтров*, *с изменением цвета подсветки* и *с разделением луча*.

Наиболее простая и распространенная технология — применение **цветных фильтров** (красный, синий, зеленый — RGB), располагаемых между датчиком и сканируемым объектом. Современная технология автоматической смены фильтров (в первых сканерах фильтры менялись вручную, сканирование производилось в три прохода) позволяет выполнять сканирование в один проход с троекратным экспонированием каждой линейки пикселов. Такой метод *прост и недорог*, поэтому применяется в большинстве современных сканеров. *Недостатком* метода является то, что светофильтры *существенно уменьшают интенсивность светового потока* (до 80–90%), что отрицательно сказывается на качестве сканируемого изображения.

Более качественное изображение можно получить, используя другой способ разделения цветов — применение нескольких **цветных источников света**. Сканирование может проходить как за три прохода — изображение сканируется полностью при свете каждого источника, так и за один — для каждой линии изображения по разу включается каждый источник света. В качестве источников света используют лампы с холодным катодом (флуоресцентные или ксеноновые) или светодиоды.

Цветная подсветка обладает следующими *недостатками*. Применение газонаполненных ламп делает конструкцию громоздкой, тяжелой и менее надежной. Использование светодиодов отрицательно сказывается на качестве цветопередачи. Дело в том, что спектры излучения светодиодов очень узкие, в результате мощность излучения в областях спектра между опорными цветами оказывается столь мала, что часть информации о цвете теряется.

Наиболее совершенный способ цветоделения — **разделение полного спектра белого цвета на хроматические составляющие**. Преимуществом разделения пучка является быстрое и качественное сканирование как цветных, так и монохромных изображений без каких-либо артефактов или искажений цвета. Однако сложная оптическая система, состоящая из нескольких зеркал и фильтров, чрезвычайно требовательна к точности позиционирования сканирующей головки, что обусловливает высокую стоимость подобного типа сканеров.

На современном рынке представлен широкий спектр сканеров различных моделей и типов. При классификации этого разнообразия обычно используют три признака: *способ размещения оригинала*, *область применения* и *тип интерфейса*.

По способу размещения оригинала различают:

- ручные сканеры;
- протяжные (роликовые) сканеры;

- планшетные сканеры;
- проекционные (книжные) сканеры;
- сканеры для прозрачных оригиналов;
- барабанные сканеры.

Ручные сканеры являются самыми компактными и недорогими. Они не имеют механического привода сканирующей головки, при сканировании пользователь перемещает ее вручную. Некоторые модели ручных сканеров оснащаются колесиками для замера перемещения сканера по поверхности, чтобы минимизировать искажения, связанные с неизбежной неравномерностью движения прибора. В наиболее совершенных моделях предусмотрена даже система обратной связи, обеспечивающая более равномерное движение.

Ширина сканируемой области ручных сканеров обычно не превышает 10–12 см, более широкие оригиналы сканируют по частям, собирая затем воедино программным путем. Основное предназначение ручных сканеров — использование совместно с портативными компьютерами (например, в публичных библиотеках). Благодаря особенностям конструкции с их помощью удобно сканировать книги, журналы, документы, а также изображения с неплоских поверхностей, при этом обеспечивается хорошее качество распознавания текстовых оригиналов. Для сканирования фотографий ручные сканеры не подходят — неизбежные геометрические искажения делают качество получаемого изображения неприемлемым.

Протяжные (роликовые) сканеры по принципу действия напоминают факс-аппараты: сканируемый оригинал загружается в щелевой податчик и протягивается сквозь сканер. Протяжные сканеры имеют ряд неоспоримых преимуществ: они так же компактны, как и ручные модели, не требуют дополнительного питания, позволяют сканировать широкие оригиналы и при этом не вносят геометрических искажений, обеспечивая высокое качество итогового изображения. Однако протяжным сканером невозможно отсканировать книгу или журнал, и, кроме того, такой сканер может повредить ценный или хрупкий оригинал.

Планшетные сканеры — наиболее распространенный и универсальный тип сканеров. Планшетный сканер оснащается планшетом для оригиналов — прозрачной стеклянной поверхностью. Под стеклянным дном планшета с помощью шагового двигателя перемещается сканирующий блок. Такая конструкция позволяет сканировать практически любые оригиналы — от толстых книг до одинарных ветхих документов. К преимуществам планшетного сканера следует также отнести возможность одновременного сканирования нескольких небольших оригиналов, а также части оригинала.

Энергопотребление системы подсветки и механизма перемещения сканирующего блока довольно велико, поэтому большинство планшетных сканеров требуют подключения дополнительного блока питания. Несмотря на высокое качество сканирования, размер сканируемого изображения всегда будет ограничен размером планшета. Кроме того, к недостаткам этого типа сканеров следует отнести

большие габариты и иногда массу. Однако недавно появились занимающие намного меньше места на столе планшетные сканеры с вертикальным расположением оригинала.

Проекционные сканеры напоминают фотоувеличитель или проекционный аппарат и предназначены для сканирования большого количества книг и журналов, а также больших оригиналов. В этих устройствах оригинал располагается на массивном постаменте, а сканирующий блок расположен неподвижно над ним. Такая конструкция позволяет сканировать книги «естественным» способом — положив обложкой вниз и перелистывая страницы. Это очень важно при оцифровке ветхих и ценных архивных документов и книг.

Сканеры, предназначенные *для сканирования прозрачных оригиналов* (*фотопленок и слайдов*), имеют конструктивную особенность — система подсветки и сканирующий блок расположены в них по разные стороны оригинала. Фотопленка или слайды, как правило, представляют собой высококачественные оригиналы с широким динамическим диапазоном цветов и яркости. Поэтому в таких сканерах не применяется светодиодная подсветка, а в качестве датчиков используются высококачественные ПЗС с высокими плотностью элементов и разрешающей способностью.

Барабанные сканеры обеспечивают высочайшее разрешение и динамический диапазон и являются очень дорогими приборами. Оригинал закрепляется на прозрачном вращающемся барабане, внутри которого размещена лампа подсветки белого цвета. Параллельно оси барабана перемещается оптический сканирующий блок на базе ФЭУ. Его характеристики таковы, что он позволяет сканировать нюансы полутонов, не различаемые человеческим глазом, а эффективное разрешение сканирования определяется только скоростью вращения барабана. Цветоделение производится путем разделения пучка с помощью нескольких диахронических зеркал.

Характеристики схожих по конструкции сканеров могут сильно различаться в зависимости от **области применения**. По этому признаку сканеры разделяются на:

- бытовые (домашние);
- профессиональные;
- сканеры документов.

Домашние сканеры представляют собой недорогие, как правило, универсальные легкие и компактные устройства. Бытовые сканеры позволяют сканировать прозрачные и непрозрачные оригиналы, причем могут это делать максимально быстро. Однако универсальность и низкая цена достигаются за счет невысокого качества сканирования.

Профессиональные сканеры, как следует из названия, предназначены для пользователей, профессионально работающих с изображениями. Устройства этого типа — от планшетных до барабанных — обеспечивают максимальное каче-

ство сканирования. Именно в таких моделях применяются системы цветоделения на основе расщепления пучка и сканирующие датчики со сложной оптической системой. Как правило, профессиональные сканеры имеют внушительные массу и габариты.

Сканеры документов — это высокопроизводительные устройства, способные быстро оцифровывать многостраничные документы. В качестве сканеров документов используются протяжные сканеры с усовершенствованными лотками или планшетные модели с автоматической подачей оригиналов.

По **типу интерфейса** сканеры различаются в зависимости от того, какая шина «малого» интерфейса используется для подключения устройства к ЭВМ. В настоящее время для этих целей используются шины SCSI, USB и FireWire. В историю отошел устаревший 25-контактный параллельный интерфейс Centronix, которым раньше оснащались почти все сканеры младшего и среднего уровня.

Несколько слов следует сказать также о **программных интерфейсах** сканеров. Широкое распространение сканеров произошло в 1980-х годах, и в это же время встал вопрос о создании единого программного стандарта для работы со сканирующими устройствами. Для этих целей было сформировано некоммерческое объединение TWAIN Working Group, в которое вошли такие компании, как Adobe, Canon, Kodak, HP, Intel и др. Разработанный стандарт, получивший название **TWAIN** (от англ. twain — «пара», имеется в виду сопряжение сканера и ЭВМ), является открытым, за его использование не нужно платить лицензионных отчислений.

В идеологии TWAIN взаимодействие программно-аппаратных средств обработки изображения разделено на четыре уровня: уровень приложения, протокольный уровень, уровни драйвера и устройства. Программный интерфейс (API) TWAIN работает на протокольном уровне, обеспечивая взаимодействие пользовательского приложения и драйвера устройства. TWAIN позволяет приложениям работать с изображениями, полученными не только со сканеров, но и с любого другого TWAIN-совместимого устройства, например цифровой фотокамеры.

Возможности принятого в 1992 г. стандарта TWAIN через некоторое время перестали быть достаточными. Поэтому компания Microsoft разработала собственный стандарт работы с устройствами-источниками изображений — **WIA** (Windows Imaging Acquisition — «интерфейс доставки изображений Windows»). WIA тесно взаимодействует с ядром ОС Windows и содержит широкий спектр системных компонентов с возможностями, превышающими рамки стандарта TWAIN.

Недостатком WIA является его ориентация на достаточно простые и однотипные устройства. Однако многие современные сканеры среднего и старшего уровня имеют множество специфических особенностей и возможностей, которые не могут быть учтены в универсальном интерфейсе. Для обеспечения работы с подобными устройствами в середине 1990-х годов компанией Pixel Translation

был разработан стандарт **ISIS** (Image and Scanner Interface Specification — «стандартный интерфейс для работы со сканерами и изображениями»).

Основное достоинство ISIS — модульная архитектура, позволяющая разработчикам и пользователям компоновать стандартизованные программные модули (блоки) в так называемые конвейеры для решения тех или иных задач. Например, конвейер обработки договора может состоять из блоков многостраничного сканирования, преобразования в определенный графический формат, сжатия и сохранения в файл. ISIS предоставляет разработчикам ПО мощную и гибко настраиваемую систему управления устройствами сканирования, а производителям сканеров — максимальную совместимость их продукции с существующим ПО.

4.5.3.4. Трехмерные сканеры

Трехмерные сканеры представляет собой устройства для ввода в ЭВМ трехмерного (объемного) изображения. По принципу работы их можно разделить на *контактные* и *бесконтактные*.

К первым относятся механические устройства, имеющие какой-либо «ощупывающий орган», координаты места соприкосновения которого с объектом оцифровываются. К этому типу принадлежат рычажные сканеры. Принцип их действия состоит в вычислении координат острия щупа, который крепится на «руке» из нескольких рычагов, соединенных шарнирами. Внутри каждого шарнира находится точный датчик угла поворота. Именно от его качества зависит то, насколько точно могут быть вычислены координаты щупа, которым прикасаются к поверхности сканируемого объекта. Примером могут служить сканеры MicroScribe фирмы Immersion.

Бесконтактные сканеры используют гораздо более широкий арсенал технических принципов, некоторые из которых являются собственной разработкой фирм-производителей. Общим в действиях таких сканеров является применение лазерного луча в качестве «ощупывающего» элемента. Дополнительно многие бесконтактные сканеры оснащены специализированной фотокамерой, которая работает синхронно со сканирующим устройством. Такой тандем дает возможность получать не только данные о рельефе поверхности объекта, но и о цвете. После съемки объекта пользователь получает уже практически готовую трехмерную модель с текстурой. Выпуском сканеров подобного рода занимается компания Cyberware.

Часто необходима максимальная оперативность в получении цифровой модели объекта. Это предполагает и минимальное время съемки, и отсутствие необходимости установки сложного и громоздкого оборудования. При этом точность и разрешение могут оказаться не главными параметрами. Для решения таких задач существует класс трехмерных сканеров, который можно назвать «трехмерные камеры». Принцип работы с ними прост — нужно установить сканирующее устройство на твердую поверхность, направить объектив на сканируемый пред-

мет, определив расстояние до него, и запустить сканер. По такому принципу работают, например, трехмерные сканеры фирмы Minolta.

Когда требуется получить цифровую модель целого помещения или очень большого объекта, ни одно из упомянутых устройств не позволит сделать это достаточно быстро и удобно. Для этого существуют специальные устройства, работающие по принципу обзорной лазерной локации. Примером может служить сканер фирмы MetrieVision. Это устройство представляет собой сканирующий модуль, размещенный на вращающемся постаменте. Такое расположение позволяет ощупывать лучом все помещение вокруг сканера. Если произвести сканирование с нескольких точек, а затем совместить полученные данные, то можно построить очень подробную и точную модель помещения, вписывающегося в сферу диаметром 48 м.

Трехмерные сканеры являются сложными оптико-механическими устройствами, требующими высокой точности исполнения, а также сопутствующего программного обеспечения, выполняющего специализированные математические расчеты во время работы сканера. Как следствие, данные изделия имеют малую серийность и высокую цену. Тем не менее существуют постоянные потребители подобной аппаратуры — группы промышленного дизайна крупных фирм, подразделения машиностроительных компаний, крупные фирмы — разработчики игр и т. п.

4.5.4. Устройства вывода информации

4.5.4.1. Матричные принтеры

Матричные принтеры получили свое название благодаря тому, что каждый выводимый на печать символ представлен в виде растровой матрицы. В настоящее время используются *игольчатые* и *линейные* принтеры.

Игольчатый матричный принтер (Dot Matrix Printer) долгое время принадлежал к стандартному устройству вывода. В то время когда струйные принтеры работали еще неудовлетворительно, а цена лазерных была слишком высока, игольчатые принтеры повсеместно использовались с домашними компьютерами. Они часто применяются и сегодня. Достоинства этих принтеров определяются в первую очередь способностью работы с любой бумагой, а также низкой стоимостью печати и возможностью одновременной печати нескольких копий.

Метод, с помощью которого игольчатый принтер переносит знаки на бумагу, в принципе очень прост. В отличие от типового принтера, в котором устанавливаются различные типы (буквы, цифры и знаки препинания), игольчатый принтер формирует знаки несколькими иголками, расположенными в головке принтера.

Бумага втягивается в принтер с помощью вала; между бумагой и головкой принтера располагается красящая лента. При ударе иголки по этой ленте на бумаге остается закрашенный след. Иголки, вертикально расположенные внутри головки, обычно приводятся в действие электромагнитами. Головка перемещается по горизонтальным направляющим с помощью шагового двигателя. За один шаг двигателя может быть напечатан один столбец растровой матрицы символа. В первых

игольчатых принтерах в головке принтера находилось 9 иголок, затем появились 18-игольчатые принтеры. В настоящее время большинство фирм-изготовителей перешли на производство 24-игольчатых принтеров.

Благодаря горизонтальному движению головки принтера и активизации отдельных иголок напечатанный знак образует как бы матрицу, причем отдельные буквы, цифры и знаки записаны в память принтера (ПЗУ) в виде бинарных кодов. Поэтому головка принтера «знает», какие иголки и в каких комбинациях необходимо активизировать, чтобы, например, создать за определенное число шагов головки некоторую букву.

Хотя наличие девяти иголок в головке принтера обеспечивает высокую скорость печати, высокого качества достичь не удается. В 24-игольчатых принтерах (сегодняшний стандарт матричных принтеров) используется технология последовательного расположения иголок в два ряда по 12 иголок. Вследствие того что иголки в соседних рядах сдвинуты по вертикали, точки на распечатке перекрываются таким образом, что их невозможно различить. Также имеется возможность прохода головки дважды для каждой строки, чтобы знаки пропечатались еще раз с небольшим смещением. Изображение буквы, возникающее таким образом, только при тщательном рассмотрении можно идентифицировать как матрицу. Поэтому такое качество печати обозначают как L, что является сокращением от Letter quality — «высокое качество». Несколько худшую по качеству печать соответственно обозначают NL (Near Letter quality — «близкое к высокому качество»). При работе в режиме L скорость печати уменьшается незначительно, так как головка печатает при движении в обоих направлениях: как слева направо, так и справа налево.

Для скоростной широкоформатной печати используются **линейно-матричные** принтеры. Принтер создает символы и графику, печатая набор точек во всю ширину печати. Каждый текстовый символ также хранится в памяти принтера в виде растровой матрицы точек. Чернильные точки наносятся линейкой маленьких молоточков, расположенных на возвратном механизме, который быстро перемещается в горизонтальной плоскости. Логические элементы принтера делят поток данных для печати на линии и столбцы. Печатающие молоточки наносят точки в определенных столбцах одновременно для всей ширины печати, перенося чернила с красящей ленты на бумагу.

Когда возвратный механизм достигает конечного положения, он меняет направление движения на противоположное, бумага подается на одну линию точек вперед и печатающие молоточки наносят на бумагу следующую строку точек. После того как линия символов напечатана, печатающие молоточки останавливаются, и бумага подается на начало первой строки точек следующей строки текста.

Поскольку линейно-матричный принтер печатает всю ширину поля печати одновременно, он является более производительным, чем матричный принтер, печатающая головка которого печатает последовательно символ за символом. Скорость печати измеряется в линиях на дюйм, а не в символах в минуту.

Много различных настроек влияют на скорость печати, одна из главных — выбранный пользователем шрифт. Символы, содержащие атрибуты — жирный шрифт, подчеркнутый шрифт, надстрочный шрифт, подстрочный шрифт, растянутый шрифт, сжатый шрифт, — сокращают скорость печати. Однако скорость печати не зависит от ширины области печати.

Способность принтера подавать бумагу вперед и назад позволяет печатать на одной линии с различной плотностью. Это очень полезно при печати форм и текста или текста с различными шрифтами. Использование различной плотности печати и обратной подачи бумаги при печати также влияет на скорость печати.

Как уже отмечалось, к преимуществам матричных принтеров относят дешевизну и высокую скорость печати при приемлемом качестве. Главным недостатком этих устройств является недопустимо высокий уровень шума. Поэтому их применение ограничивается помещениями, где шум имеет второстепенное значение — цеховыми, торговыми залами и т.п.

4.5.4.2. Струйные принтеры

Основной принцип работы **струйных принтеров** заключается в использовании тонких, как волос, сопел, которые находятся в головке принтера. В этой головке устанавливается резервуар с жидкими чернилами, которые через сопла, как микрочастицы, переносятся на материал носителя. Число сопел (от 16 до 64) зависит от модели принтера и изготовителя. Некоторые модели имеют гораздо большее число сопел. Например, головка HP DeskJet 1600 имеет 300 сопел для черных чернил и 416 для цветных.

Для подачи чернил используются два метода:

1) головка принтера объединена с резервуаром для чернил; замена резервуара с чернилами одновременно связана с заменой головки;

2) используется отдельный резервуар, который через систему капилляров обеспечивает чернилами головку принтера.

В своей работе струйные принтеры могут использовать следующие методы:

1) пьезоэлектрический метод;

2) метод газовых пузырей;

3) метод «drop-on-demand».

Для реализации **пьезоэлектрического метода** в каждое сопло устанавливается плоский пьезокристалл, связанный с диафрагмой. Под воздействием электрического поля происходит деформация пьезоэлемента. При печати находящийся в трубке пьезоэлемент, сжимая и разжимая трубку, наполняет капиллярную систему чернилами. Чернила, которые отжимаются назад, перетекают обратно в резервуар, а чернила, которые «выдавились» наружу, оставляют на бумаге точку.

Метод газовых пузырей является термическим и больше известен под названием **bubblejet** («инжектируемые пузырьки»). При использовании этого метода каждое сопло оборудуется нагревательным элементом, который при пропуска-

нии через него тока за несколько микросекунд нагревается до температуры около 500° С. Возникающие при резком нагревании газовые пузыри (bubbles) стараются вытолкнуть через выходное отверстие сопла каплю жидких чернил, которая переносится на бумагу. При отключении тока нагревательный элемент остывает, паровой пузырь уменьшается и через входное отверстие поступает новая порция чернил.

Метод drop-on-demand разработан фирмой Hewlett-Packard. Так же как в методе газовых пузырей, здесь для подачи чернил из резервуара на бумагу применяется нагревательный элемент. Однако в методе drop-on-demand для подачи чернил дополнительно используется специальный механизм. Технология drop-on-demand обеспечивает наиболее быстрое впрыскивание чернил, что позволяет существенно повысить качество и скорость печати.

Благодаря тому, что в механизмах печати, реализованных с использованием метода газовых пузырей, меньше конструктивных элементов, такие принтеры надежней в работе, и срок их эксплуатации более продолжителен. Кроме того, использование этой технологии позволяет добиться наиболее высокой разрешающей способности принтеров. Обладая высоким качеством прорисовки линий, данный метод имеет недостаток: области сплошного заполнения получаются несколько расплывчатыми. Применение метода газовых пузырей целесообразно при необходимости печати графиков, гистограмм и т. п., тогда как печать полутоновых графических изображений получается более качественной при использовании метода drop-on-demand.

Скорость печати струйного принтера зависит от типа (цветная, черно-белая) и качества печати, а также стоимости принтера. В целом струйные принтеры значительно медленнее лазерных собратьев. Кроме этого, высохшие чернила подавляющего большинства струйных принтеров легко размываются водой.

4.5.4.3. Лазерные принтеры

Несмотря на сильную конкуренцию со стороны струйных принтеров, черно-белые **лазерные принтеры** на сегодняшний день являются основным средством печати, благодаря следующим преимуществам:

- высокие качество и скорость печати;
- низкая стоимость одного отпечатка;
- отпечатки устойчивы к воздействию влаги.

При этом черно-белые лазерные принтеры имеют более высокую цену, чем цветные струйные, а цветные лазерные принтеры — значительно более высокую цену. Таким образом, для получения высококачественной черно-белой распечатки следует отдавать предпочтение лазерному принтеру по сравнению со струйным.

Большинством изготовителей лазерных принтеров используется механизм печати, который применяется в копировальных аппаратах (ксероксах). Принтеры HP и QMS, например, используют механизм печати ксероксов фирмы Canon.

Важнейшим конструктивным элементом лазерного принтера (рис. 4.29) является вращающийся **фотобарабан**, с помощью которого производится перенос изображения на бумагу. Барабан представляет собой металлический цилиндр, покрытый тонкой пленкой фотопроводящего полупроводника. Обычно в качестве такого полупроводника используется оксид цинка. По поверхности барабана равномерно распределяется статический заряд. Для этого служит тонкая проволока или сетка, называемая коронирующим проводом. На этот провод подается высокое напряжение, вызывающее возникновение вокруг него светящейся ионизированной области, называемой короной.

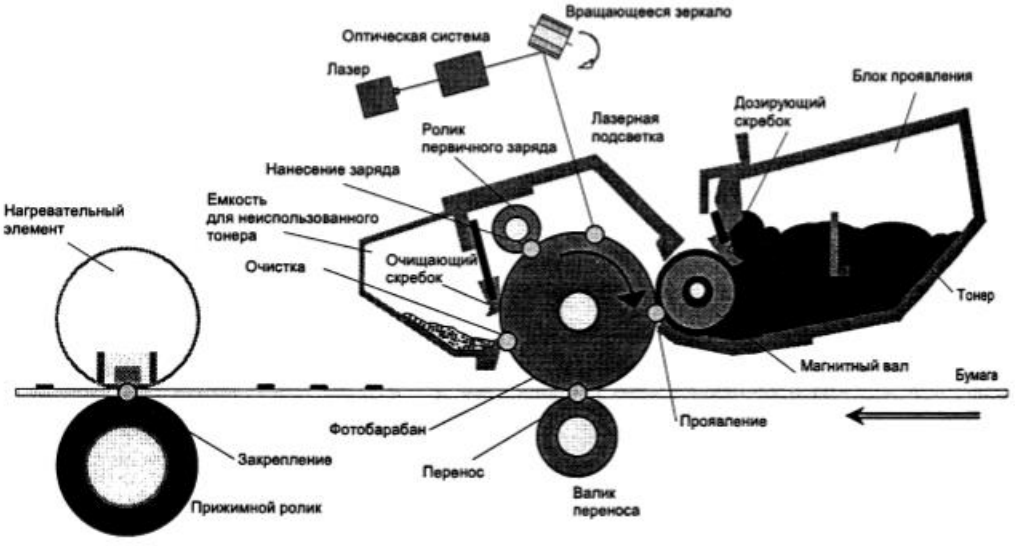


Рис. 4.29. Устройство лазерного принтера

Лазер, управляемый микроконтроллером, генерирует тонкий световой луч, отражающийся от **вращающегося зеркала**. Этот луч, приходя на барабан, изменяет его электрический заряд в точке прикосновения. Для некоторых типов принтеров потенциал поверхности барабана уменьшается от –900 до –200 В. Таким образом, на барабане возникает скрытая копия изображения.

На следующем рабочем шаге на фотонаборный барабан наносится **тонер** — мельчайшая красящая пыль. Под действием статического заряда мелкие частицы легко притягиваются к поверхности барабана в точках, подвергшихся экспозиции, и формируют изображение.

Бумага втягивается из подающего лотка и с помощью системы валиков перемещается к барабану. Перед самым барабаном бумаге сообщается статический заряд. Затем бумага соприкасается с барабаном и притягивает благодаря своему заряду частички тонера от барабана.

Закрепление изображения на бумаге выполняется сдавливанием листа с тонером между двумя валиками блока термического закрепления. Верхний валик нагревается до высокой (100–300°С) температуры и расплавляет частицы тонера. Благодаря обеспечиваемому нижним прижимным валиком давлению расплавленный тонер проникает в структуру бумаги, образуя стойкое изображение.

Обычно цветное изображение формируется при печати наложением друг на друга трех основных цветов: *циан* (cyan), *пурпурный* (magenta) и *желтый* (yellow). Хотя теоретически наложение этих трех цветов должно в итоге давать черный цвет, на практике в большинстве случаев получается серый или коричневый, и поэтому в качестве четвертого основного цвета добавляют *черный* (black). На основании этого такую цветовую модель называют *CMYK* (Cyan-Magenta-Yellow-blacK).

В цветном лазерном принтере изображение формируется на светочувствительной фотоприемной ленте последовательно для каждого цвета (cyan, magenta, yellow, black). Лист печатается за четыре прохода, что, естественно, сказывается на скорости печати. Имеются четыре емкости для тонеров и от двух до четырех узлов проявления. Принтеры этого класса оборудованы большим объемом памяти, процессором и, как правило, собственным винчестером. На винчестере располагаются разнообразные шрифты и специальные программы, которые управляют работой, контролируют состояние и оптимизируют производительность принтера. В результате цветные лазерные принтеры являются весьма габаритными и массивными аппаратами.

Разрешение лазерного принтера по горизонтали и по вертикали определяется различными факторами. Вертикальное разрешение соответствует шагу барабана и для большинства принтеров составляет 1/600 дюйма (для более дешевых — 1/300 дюйма). Горизонтальное разрешение определяется числом точек в одной строке и ограничено точностью наведения лазерного луча. Поэтому многие модели принтеров имеют «несимметричное разрешение», равное, например, 1200 × 600 dpi, когда точность перемещения лазерного луча составляет 1/1200 дюйма, но шаг барабана по-прежнему 1/600 дюйма. Воспроизводимое изображение разбивается при этом не на квадратики, а на прямоугольники со сторонами 1/600 и 1/1200 дюйма. Так как луч лазера может перемещаться не только по горизонтали, но и по вертикали, то он способен поставить точку либо в верхней, либо в нижней части прямоугольника. В этом случае говорится об алгоритмическом разрешении. Очевидно, что алгоритмическое высокое разрешение заменяет реальное лишь отчасти. Оно позволяет сделать края изображений более гладкими.

Для передачи полутонов изображение принято разбивать на несколько ячеек. Например, для принтеров с разрешением 300 × 300 dpi часто применяется квадратная ячейка из 25 точек размером 0,42 × 0,42 мм (длина стороны 1/60 дюйма), со сторонами, повернутыми на 45° относительно вертикали. При этом возможна передача 26 оттенков серого (от 0 до 25 точек в ячейке). В высококачественных принтерах такая ячейка состоит из 128 точек (например, принтеры фирмы Optra)

и тоже имеет вид квадрата, развернутого на 45°. При разрешении 1200 × 1200 dpi его размер составляет 0,25 × 0,25 мм. Качество изображения улучшается не только потому, что меньше размер ячейки, но и из-за увеличения числа оттенков серого до 129.

4.5.4.4. Термические принтеры

Для получения цветного изображения качества, близкого к фотографическому, или изготовления допечатных цветных проб используются **термические принтеры** или, как их еще называют, **цветные принтеры высокого класса**.

Различают три технологии цветной термопечати:

1) струйный перенос расплавленного красителя (термопластичная печать);

2) контактный перенос расплавленного красителя (термовосковая печать);

3) термоперенос красителя (сублимационная печать).

Общим для последних двух технологий является нагрев красителя и перенос его на бумагу (пленку) в жидкой или газообразной фазе. Многоцветный краситель, как правило, наносится на тонкую лавсановую пленку (толщиной 5 мкм). Пленка перемещается с помощью лентопротяжного механизма, который конструктивно схож с аналогичным узлом игольчатого принтера. Матрица нагревательных элементов за 3–4 прохода формирует цветное изображение.

Термовосковые принтеры переносят краситель, растворенный в воске, на бумагу, нагревая ленту с цветным воском. Как правило, для подобных принтеров необходима бумага со специальным покрытием. Термовосковые принтеры обычно используются там, где требуется высокое качество цветной печати.

Для печати изображения, почти не отличающегося от фотографии, и изготовления допечатных проб лучше всего использовать сублимационные принтеры. По принципу работы они аналогичны термовосковым, но переносят с ленты на бумагу только краситель (не имеющий восковой основы).

Принтеры, использующие струйный перенос расплавленного красителя, называют еще восковыми принтерами с твердым красителем. При печати блоки цветного воска расплавляются и выбрызгиваются на носитель, создавая краски насыщенного цвета на любой поверхности. Полученные таким образом «фотографии» выглядят слегка зернистыми, но удовлетворяют всем критериям фотографического качества. Этот принтер не годится для изготовления диапозитивов, поскольку капли воска после высыхания имеют полусферическую форму и создают сферический эффект.

Имеются термические принтеры, которые совмещают в себе технологию сублимационной и термовосковой печати. Такие принтеры позволяют печатать на одном устройстве как черновые, так и чистовые оттиски.

4.5.4.5. Мониторы (дисплеи)

Под **мониторами (дисплеями)** понимаются устройства визуального отображения информации в виде текста, графического рисунка, видео и т.п. На заре соз-

дания ЭВМ вплоть до 50-х годов XX в. информация выводилась на печатающие устройства и в некоторых случаях на осциллографы.

В 1950 г. в Кембриджском университете (Англия) была использована для вывода информации **электронно-лучевая трубка** (**ЭЛТ** или **CRT — Cathode Ray Tube**) осциллографа. Технология получения изображения была разработана немецким ученым Фердинандом Брауном в 1897 г.

Реальный прорыв в использовании дисплеев для вывода информации из ЭВМ связан с военным проектом «Вихрь» (США). В этом проекте компьютер использовался для фиксации информации о вторжении самолетов в воздушное пространство США. Первая демонстрация системы «Вихрь» состоялась 20 апреля 1951 г.

Используемые в то время мониторы (дисплеи) были **векторными**. Электронный пучок создавал на экране линии, перемещаясь непосредственно от одного набора координат к другому. Позднее появились мониторы с **растровым сканированием**. В этих дисплеях электронный пучок сканирует экран слева направо и сверху вниз, пробегая каждый раз всю поверхность экрана.

Следующим этапом было появление цветных дисплеев. В этом случае для появления изображения уже использовались три электронных пучка, каждый из которых высвечивал определенные точки на экране.

Развиваясь в течение десятилетий, технология изготовления CRT-мониторов в настоящее время достигла своей вершины. С годами эти дисплеи приобрели плоский экран с большой диагональю, высокое разрешение, идеальную цветопередачу, множество электронных настроек, позволяющих корректировать многочисленные параметры изображения (яркость, контрастность, геометрию, цветовую температуру и т. п.). Вместе с тем CRT-мониторы обладают существенными недостатками:

- большие габариты и масса;
- постепенное выгорание люминофорного слоя, что со временем вызывает падение яркости дисплея;
- значительное электромагнитное излучение, вредное для здоровья;
- трудность получения абсолютно неподвижного изображения («моргание картинки»), что вызывает утомление глаз при длительной работе с документами;
- невозможность обеспечения равномерно четкого изображения по всему экрану (в центре фокусировка электронного пучка выше, чем по краям).

Недостатки CRT-мониторов, в первую очередь габариты и масса, предопределили их постепенный уход с рынка. Их место занимают **жидкокристаллические дисплеи** (**ЖК-дисплеи**, **LCD — Liquid Crystal Display**) с тонким экраном. Поскольку в ближайшие годы ЖК-мониторы будут занимать основную долю рынка, рассмотрим их строение и характеристики подробнее.

Главной составляющей ЖК-монитора является **жидкокристаллическая панель**. В ее основе лежит свойство одного из типов ЖК-веществ (*нематиков*) поворачивать плоскость поляризации проходящего через него света. Угол поворота за-

висит от положения кристалла относительно падающего пучка света, а положение кристалла, в свою очередь, можно менять с помощью электрического поля.

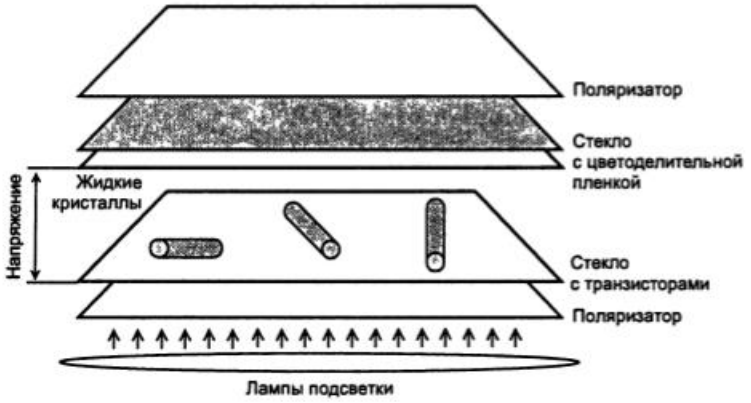


Рис. 4.30. Структура ЖК-панели

В простейшем случае панель (рис. 4.30) представляет собой две стеклянные пластины с нанесенными на них прозрачными электродами, между которыми находятся жидкие кристаллы. С обеих сторон эта конструкция закрывается двумя поляризующими пленками. Допустим, что обе пленки имеют горизонтальную поляризацию, а жидкие кристаллы расположены так, что в отсутствие электрического поля направление поляризации света не меняют. Тогда пучок света, направленный на одну сторону панели, будет горизонтально поляризован первым поляризатором и без изменений пройдет слой жидких кристаллов. Так как у второго поляризатора направление поляризации такое же, как у направляемого пучка света, пучок выйдет наружу с другой стороны панели. Поскольку к направлению поляризации света человеческий глаз не чувствителен, панель будет выглядеть просто прозрачной.

Если же, подав на электроды напряжение, изменить положение жидких кристаллов таким образом, что они будут поворачивать плоскость поляризации проходящего через них света на 90°, то панель станет непрозрачной. Свет будет полностью поглощаться вторым поляризатором, поскольку его направление поляризации окажется перпендикулярным к направлению поляризации света. Соответственно поворачивая кристаллы на промежуточные углы путем изменения напряжения на электродах, можно плавно регулировать прозрачность панели. При этом один кристалл будет определять одну точку на экране — **пиксел**.

Сама по себе ЖК-панель ничего не излучает, она лишь меняет интенсивность проходящего через нее света, поэтому для работы панели требуется внешняя подсветка. В ЖК-дисплеях применяются, как правило, либо ртутные флюоресцентные лампы с холодным катодом (CCFL — Cold Cathode Fluorescent Lamp), пи-

тающиеся от высоковольтного преобразователя, либо ксеноновые лампы низкого давления.

Для получения цветного изображения каждый пиксел панели разбивается на три независимых **субпиксела**, и на него накладывается так называемая **цветоделительная маска**. Последняя представляет собой фильтр, окрашивающий проходящий через каждый из субпикселов свет в один из основных цветов — *красный*, *синий* или *зеленый*. Таким образом, при освещении панели белым светом получаются привычные для любого монитора RGB-триады.

Описанная выше технология ЖК-дисплеев в виде двух стеклянных пластин с нанесенными на стекло электродами исторически была первой. Панели, построенные по этой технологии, называются **пассивными матрицами** и обладают очень низкими углами обзора и крайне большим временем реакции.

В **активно матричных панелях** каждый субпиксел имеет свой собственный управляющий транзистор и снабжается параллельно включенным конденсатором, позволяющим поддерживать постоянное напряжение на пикселе и полностью исключить мерцание панели. Из-за требования к прозрачности матрицы в целом транзисторы имеют толщину менее 0,1 мкм — так называемые **тонкопленочные транзисторы** (**Thin Film Transistors**, **TFT**). В настоящее время аббревиатура TFT является синонимом понятия «активная матрица».

К основным параметрам мониторов следует отнести:

- размер диагонали;
- габариты и массу;
- время отклика или послесвечения;
- углы обзора;
- яркость и контрастность;
- различные аспекты цветопередачи.

Массогабаритные характеристики, хотя и очень важны, являются тривиальными и не определяют качество изображения.

Понятие «**время отклика**» появилось вместе с ЖК-панелями, его не следует путать со «**временем послесвечения**», которое актуально для CRT-мониторов.

В электронно-лучевой трубке луч проходит через каждую точку экрана с частотой кадровой развертки. При попадании очередной точки люминофора в зону луча она практически мгновенно начинает светиться, а после ухода луча гаснет, но не мгновенно, а плавно в течение нескольких миллисекунд. Движущийся на черном фоне белый квадрат на CRT-мониторе будет иметь абсолютно четкую переднюю границу и слабый белый «хвост» послесвечения. При этом яркость «хвоста» будет спадать сначала очень быстро (т. е. задняя граница квадрата тоже будет почти четкой), а слабый след может иметь заметную длину.

На ЖК-мониторе движущиеся изображения выглядят иначе. Из-за сравнительно большой вязкости жидких кристаллов с момента изменения электрического поля до момента окончания поворота кристалла может пройти от единиц до

десятков миллисекунд. Движущийся на черном фоне белый квадрат будет иметь нечеткую переднюю границу из-за ненулевого времени зажигания пиксела и такую же нечеткую заднюю границу из-за ненулевого времени гашения. В отличие от мониторов на ЭЛТ, за квадратом практически не будет слабого светлого следа послесвечения.

Согласно стандарту ISO 13406-2 **время отклика** ЖК-панели определяется как суммарное время зажигания и гашения пиксела, причем измеряется не полное время, а время изменения яркости от 10% от максимальной до 90% и обратно. Заметим, что пиксел при измерении переходит между двумя крайними состояниями, а переходы между промежуточными состояниями, т. е. между оттенками серого, не учитываются. Если бы скорость поворота кристалла была постоянной, тогда переходы с черного на белый были бы самыми длительными, и время отклика в понимании ISO 13406-2 характеризовало бы матрицу с наихудшей стороны. Однако кристаллы управляются электрическим полем, и для того чтобы повернуть кристалл на меньший угол, к ячейке ЖК-панели требуется приложить меньшее напряжение. Это приведет к меньшей величине электрического поля между электродами и соответственно меньшей скорости поворота кристалла. В связи с этим практически для всех типов матриц время переключения с черного на белый и обратно оказывается минимальным из достижимых. Переключение между любыми другими промежуточными состояниями будет занимать в лучшем случае такое же время, но в абсолютном большинстве случаев — существенно большее, для некоторых типов матриц разница может превышать 5 раз, что необходимо учитывать пользователю монитора.

Время отклика (послесвечения) некритично при работе с неподвижными объектами (текстовыми документами, чертежами, фотографией), но имеет крайне важное значение при отображении динамических объектов (видео, игровых сцен и т. п.).

Такой параметр, как **углы обзора**, имеет смысл исключительно для ЖК-дисплеев. Угол, под которым световой пучок падает на жидкий кристалл, с одной стороны, определяет угол поворота плоскости поляризации света, с другой — зависит от угла, под которым пользователь смотрит на экран. При отклонении в сторону матрица начинает терять контрастность — черный цвет становится светлее, а белый, напротив, темнее. Текущий стандарт определяет углы обзора монитора как углы относительно перпендикуляра к экрану, под которыми контрастность в центре экрана снижается до 10 : 1. При этом указываются не четыре, а только два угла обзора — вертикальные и горизонтальные углы попарно суммируются.

Отметим мягкость и неинформативность стандарта. Во-первых, на практике контрастность 10:1 означает, что с экрана можно читать текст, но никак не изучать фотографии или смотреть фильм — глаз воспринимает такое изображение как весьма блеклое. Во-вторых, измерения проводятся лишь в центре экрана, хотя очевидно, что края экрана будут видны под другим углом. В-третьих, принима-

ется во внимание только изменение контрастности, но не цветовые искажения, которые характерны для многих типов матриц. В-четвертых, не учитывается возможная несимметричность углов обзора, так как вертикальные и горизонтальные углы попарно суммируются. Таким образом, реальные углы обзора, при которых глаз не будет замечать существенных искажений изображения, как правило, оказываются намного меньше указанных производителем паспортных значений.

Яркость монитора традиционно определяется как измеряемая в центре экрана яркость чистого белого цвета в канделах на квадратный метр (кд/м^2) при цветовой температуре 6500°K. Максимальная яркость современных мониторов варьируется в зависимости от модели от 185 до 300 кд/м^2.

Контрастность определяется как отношение яркости белого цвета к яркости черного цвета. Заметим, что в отличие от мониторов на ЭЛТ к ЖК-мониторам понятие «черный цвет» может быть применено только условно, так как в силу своего принципа работы некоторый процент света они пропускают всегда. Поэтому «выключенный» пиксел будет на самом деле не черным, а темно-серым. Это является наиболее *серьезным недостатком ЖК-дисплеев*. Контрастность CRT-мониторов может быть выше в 4–5 раз.

Цветопередача характеризует достоверность отображения монитором цветовой гаммы и зависит от многих параметров, среди которых выделяют три наиболее важных: *гамма-коррекция*, *цветовая температура* и *разрядность матрицы*.

Цветопередача монитора описывается так называемыми цветовыми кривыми, т. е. зависимостью между входным (заданным видеокартой цветом) и выходным сигналом. Эта зависимость имеет вид:

$$O = I^{\gamma},$$

где I — входной сигнал; O — выходной; γ — некоторое число, причем максимальный входной сигнал принят равным единице, а минимальной — нулю. Данное преобразование (**гамма-коррекция**) позволяет улучшить передачу темных оттенков изображения. Гамма-коррекция осуществляется дважды с противоположным знаком: первый раз на стадии подготовки изображения, второй — при его отображении. В результате на экране монитора оригинал отображается так, как он выглядел до первой коррекции.

Исторически сложилось, что Apple-совместимые компьютеры использовали $\gamma = 1{,}8$, обеспечивая наиболее точную цветопередачу. В массовых ЭВМ на платформе Intel/Windows стандартным значением γ было 2,5. Это связано с тем, что электронно-лучевая трубка в силу особенностей работы имеет γ от 2,45 до 2,55. В настоящее время разработан стандарт sRGB (A Standard Default Color Space for the Internet), согласно которому значение γ должно составлять 2,2, что позволяет с минимальными искажениями просматривать изображения, скорректированные под значения γ, равные 1,8 или 2,5.

Для CRT-мониторов передаточная характеристика с $\gamma = 2{,}5$ уже есть в силу самого их устройства, причем скорректировать ее под другие числа γ не пред-

ставляет особой сложности. В ЖК-мониторах электронике дисплея приходится искусственно выстраивать кривую гамма-коррекции, при этом результат довольно часто получается далеким от идеала, так как кривая зависит от входного сигнала и не совпадает с теоретической. На практике это означает, что если реальная кривая проходит выше идеальной на темных тонах и ниже на светлых, то монитор будет отображать темные оттенки светлее, чем они есть на самом деле, а светлые — наоборот, темнее.

Еще один важный параметр изображения с точки зрения цветопередачи — **цветовая температура**. Она определяет тональность изображения на экране монитора — чем ниже температура, тем «теплее» цвета (человек воспринимает как более холодный спектр излучения тела, которое на самом деле более горячее). Для человеческого глаза нет понятия абсолютно белого цвета — глаз адаптируется под различные условия освещения и в зависимости от них может воспринимать как белый несколько различные цвета. Для компенсации этого эффекта в мониторах предусмотрена регулировка цветовой температуры, позволяющая настроить монитор так, чтобы в конкретных условиях белый цвет на нем выглядел именно белым, а не желтоватым и не голубоватым.

Цветовая температура измеряется в градусах Кельвина (°K) и равна температуре абсолютно черного тела, излучающего такой же спектр. Существуют два наиболее распространенных значения — 5500°K, использующееся обычно при печати и в фотографии, и 6500°K, использующееся при обработке изображений на мониторе. Для сравнения заметим, что цветовая температура 6000°K соответствует яркому солнечному свету при безоблачном небе, а температуру 6500–7000°K имеет небо с легкой облачностью. Предусмотрен также вариант установки температуры 9300°K, дающий легкий синеватый оттенок (природный аналог — цветовая температура легкой тени в ясный день). Стандарт для компьютерных мониторов согласно sRGB — цветовая температура 6500°K.

Такая характеристика, как **разрядность матриц**, применима только к ЖК-мониторам. Стандартным значением разрядности для современных графических адаптеров и мониторов является 24-битный цвет (на каждый из трех компонентов приходится по 8 бит), обеспечивающий передачу 16,7 млн различных цветов. Для аналоговых CRT-мониторов понятие «количество цветов» напрямую неприменимо. Напротив, для ЖК-дисплеев, на матрицы которых подается цифровой сигнал, понятие разрядности имеет точно такое же значение, как и для графических адаптеров.

Абсолютное большинство современных матриц имеют 18-битную цветопередачу и могут отображать не более 262 тысяч цветов. Для достижения требуемого количества цветов в ЖК-панелях применяется технология покадровой смены цвета. Если необходимо вывести точку отсутствующего у матрицы цвета, то вместо нее поочередно выводятся точки двух ближайших имеющихся цветов, один немного темнее нужного, другой — светлее. Цвета меняются с частотой кадровой

развертки, благодаря чему глаз видит некий «усредненный» цвет, весьма близкий к тому, который необходимо вывести на экран. Данная технология обычно встречается под названием **FRC** — Frame Rate Control.

В настоящее время все дисплеи принято классифицировать по следующим признакам:

- схема формирования изображения;
- размер;
- тип матрицы (для ЖК-дисплеев).

На рис. 4.31 представлена классификация дисплеев по указанным признакам.

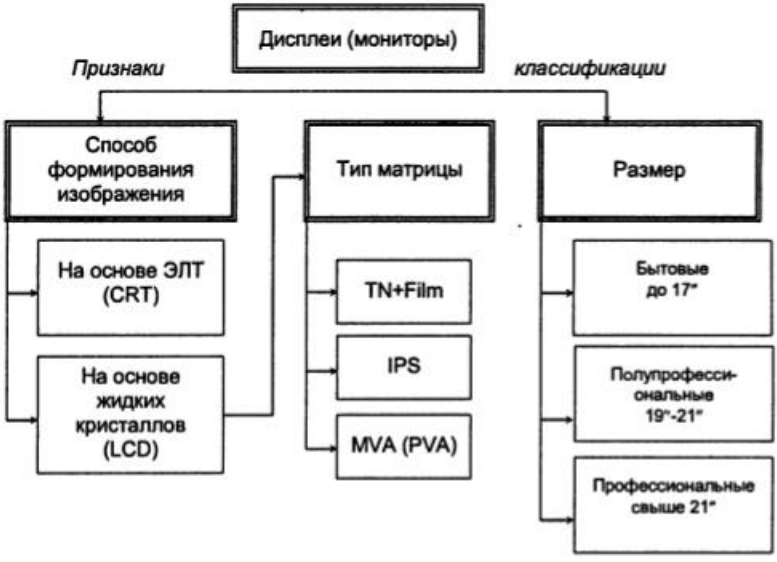


Рис. 4.31. Классификация мониторов

Отметим, что наиболее распространены три типа ЖК-матриц, отличающихся друг от друга технологией изготовления.

Наиболее распространенной и дешевой технологией является **TN + Film** (**Twisted Nematics**). TN-технология была исторически первой. Для нее, помимо низкой цены, были характерны весьма посредственные параметры, главным образом низкая контрастность и малые углы обзора. Данные параметры удалось значительно улучшить за счет нанесения на поверхность экрана полимерной пленки (film) с большим показателем преломления. После этого технология и приобрела название TN + Film. Для ЖК-матриц этого типа характерно низкое качество статического изображения (низкая контрастность, посредственная цветопередача и т. п.). Однако данные матрицы имеют малое время отклика, что делает их незаменимыми для игр или просмотра видео. В сочетании с низкой ценой этого оказывается достаточно для лидерства на рынке.

Технология **IPS (In-Plane Switching)** применяется в мониторах с большой диагональю. Главное их достоинство — отличная цветопередача, но недостатками являются низкая контрастность, большое время отклика, высокая цена. IPS-мониторы используются в сфере профессиональной графики.

Технология **MVA (Multidomain Vertical Alignment)** в настоящее время не используется, за исключением ее аналога от компании Samsung — технологии **PVA (Patterned Vertical Alignment)**. Технология характеризуется большими углами обзора без заметных цветовых искажений. Помимо этого, данные матрицы обладают отличной контрастностью и очень хорошей цветопередачей. Единственным серьезным их недостатком является большое время отклика.

К компьютеру монитор можно подключать двумя способами: сигнальным (аналоговым) и цифровым. Для настройки параметров дисплея (яркость, контрастность и др.) используют аналоговые системы управления (ручки, движки, потенциометры и т. п.) и цифровые (кнопки, экранное меню, управление через компьютер), последние являются преобладающими в современных мониторах.

4.5.4.6. Видеоадаптеры

Текстовый режим и двумерная графика.

Для того чтобы текстовая или графическая информация появилась на экране монитора, необходимо специальное устройство — **видеоадаптер (дисплейный адаптер, видеокарта)**. За долгие годы своего развития (начиная с 1950-х годов) дисплейные адаптеры прошли путь от простейших устройств, способных отображать на экране лишь текстовую информацию, до мощных процессоров трехмерной графики.

Изображение на экране может быть сформировано одним из двух методов — *векторным* или *растровым*. При **векторном** способе изображение формируется при перемещении луча по экрану в соответствии с координатами строящихся элементов изображения. **Растровый** метод подразумевает представление изображения в виде матрицы точек. Изображение на экране формируется при перемещении луча в соответствии с разверткой слева направо по строке и сверху вниз по столбцам матрицы с подсветкой требуемых точек.

История развития машинной графики выходит за рамки нашего учебника. Отметим лишь: в процессе эволюции графических систем ЭВМ векторный метод уступил растровому, который сегодня применяется повсеместно. Первым общепринятым растровым графическим стандартом стал разработанный фирмой IBM стандарт **VGA (Video Graphic Array** — «графическая видеоматрица»). Данный стандарт поддерживается всеми видеоадаптерами и используется по умолчанию при установке практически всех операционных систем.

Современный видеоадаптер способен работать в **текстовом** и нескольких **графических** режимах. Дальнейшее объяснение принципов работы видеоадаптеров мы будем вести в предположении, что в качестве дисплея используется CRT-монитор.

Такой подход вполне оправдан, так как изначально механизмы функционирования дисплейных адаптеров разрабатывались с ориентацией именно на CRT-мониторы. Появившиеся значительно позже ЖК-дисплеи были вынуждены «подстраиваться» под уже существующую технологию.

В любом **графическом режиме** видеоадаптера имеется возможность индивидуального управления свечением каждой точки экрана монитора независимо от состояния остальных. Каждой точке экрана — **пикселу** — соответствует ячейка специальной памяти, которая сканируется схемами адаптера синхронно с движением луча монитора. Эта постоянно циклически сканируемая (с кадровой частотой) память называется **видеопамятью (Video Memory)**. Основой видеопамяти являются микросхемы динамических ОЗУ. Процесс постоянного сканирования видеопамяти называется **регенерацией изображения**, и этого же сканирования оказывается достаточно для регенерации информации в динамических ОЗУ. Для программно-управляемого построения изображений к видеопамяти обеспечивается доступ и со стороны системной магистрали ЭВМ, причем как по записи, так и по чтению. С этой целью, видеопамять, физически расположенная на адаптере, логически включается в адресное пространство основной памяти ЭВМ. Другими словами, обращение к видеопамяти со стороны процессора выглядит, как запись или чтение основной памяти по определенным адресам.

Количество бит видеопамяти, отводимое на каждый пиксел, определяет возможное число состояний пиксела — цветов, градаций яркости или иных атрибутов (например, мерцание). Так, при одном бите на пиксел возможны лишь два состояния — «светится» или «не светится». Два бита на пиксел позволяют иметь одновременно четыре цвета на экране. Современные видеоадаптеры поддерживают, как минимум, следующие режимы:

- режим **VGA** — 8 бит на пиксел, 256 цветов;
- режим **High Color** («качественный цвет») — 16 бит на пиксел, 65536 цветов;
- режим **True Color** («верный цвет») — 24 (32) бита на пиксел, 16,7 млн (4,3 млрд) цветов.

Биты цвета по возможности равномерно распределяются между базисными цветами RGB (Red-Green-Blue — «красный-зеленый-синий»).

Логически видеопамять может быть организована по-разному в зависимости от количества бит на пиксел, но в любом случае имеет место *отображение матрицы пикселов экрана на биты видеопамяти* (Bit Mapping). Растровый формат хранения изображений, при котором биты так или иначе отображают пикселы, называется **битовой картой** (Bit-Map). Объем видеопамяти (в битах), требуемый для хранения образа экрана, определяется как произведение **разрешения экрана** (количества пикселов в строке, умноженного на количество строк) на количество бит на пиксел. Так, для режима с разрешением 800 × 600 и 256 цветов требует-

ся 480 000 бит или около 469 Кбайт, а для режима 1024x768 True Color — 2,25 Мбайт.

Формирование битовой карты изображения в видеопамяти дисплейного адаптера производится под управлением программы, исполняемой центральным процессором. При решении задачи формирования изображения требуется пересылка большого объема информации в видеопамять, а для многих построений — еще и чтение видеопамяти со стороны процессора. Поэтому канал связи процессора с видеопамятью представляет собой «узкое горло», через которое транслируется интенсивный поток данных, причем, чем более высокое разрешение экрана и чем больше цветов (бит на пиксел), тем этот поток должен быть интенсивнее.

Для решения проблемы «узкого горла» исторически применяются несколько подходов. Во-первых, повышают быстродействие видеопамяти. Во-вторых, расширяют разрядность шин видеоадаптера и применяют высокопроизводительные шины (например, AGP). Расширение разрядности позволяет за один цикл обращения передать большее количество бит данных и повысить производительность. В-третьих, повысить скорость видеопостроений можно применением кэширования видеопамяти. В этом случае при записи в область видеопамяти данные будут записаны как в видеопамять, так и в основную память (или кэш), при считывании из этой области обращение будет только к ОП. В-четвертых, можно принципиально сократить объем информации, передаваемой видеоадаптеру, за счет передачи ему части функций по построению изображения.

В последнем случае видеоадаптер оснащается специальным **графическим процессором**, способным формировать растровое изображение в видеопамяти по командам, полученным от центрального процессора. Команды ориентируются на наиболее часто используемые методы описания изображений, которые строятся из отдельных графических элементов более высокого уровня, чем пикселы (например, команды рисования прямоугольников, дуг и других элементов плоских изображений — так называемых элементов **2D-графики** — «двумерной графики»). Кроме этого, современные графические процессоры берут на себя и многие функции построения трехмерных изображений (или **3D-графики** — «трехмерной графики»).

По данным видеопамяти изображение (кадр) на экране монитора формируется с помощью трех (по одному на каждый базисный цвет) цифро-аналоговых преобразователей (DAC — Digital to Analog Converter). Поэтому часто отображаемую видеопамять называют **кадровым буфером**. Поскольку видеопамять основана на динамических ОЗУ с произвольным доступом (RAM), модуль, состоящий из кадрового буфера и цифро-аналогового преобразователя, получил название **RAMDAC**. Как и у любого цифрового устройства, скорость работы RAMDAC задается тактовой частотой видеоадаптера. При каждом обращении к кадровому буферу DAC считывает информацию об одной точке изображения. Зная текущее разрешение экрана и тактовую частоту адаптера, легко получить значение частоты обнов-

ления изображения на экране монитора. Заметим, что минимальная безопасная для человеческого глаза частота обновления экрана составляет 72 Гц.

В текстовом режиме формирование изображения происходит несколько иначе. Если в графическом режиме каждой точке экрана соответствует своя ячейка видеопамяти, то в текстовом режиме ячейка видеопамяти хранит информацию о символе, занимающем на экране знакоместо определенного формата. Знакоместо представляет собой матрицу точек, в которой может быть отображен один из символов определенного набора. В ячейке видеопамяти хранится код символа, определяющий его индекс в таблице символов, и атрибуты символа, определяющие вид его отображения. К атрибутам относятся цвет фона, цвет символа, инверсия, мигание и подчеркивание символа. Поскольку изначально в дисплеях использовали только алфавитно-цифровые символы, такой режим работы иногда сокращенно называют **AN (Alpha-Numerical — «алфавитно-цифровой»)**.

В текстовом режиме экран организуется в виде матрицы знакомест, образованной горизонтальными линиями (row) и вертикальными колонками (column). Этой матрице соответствует аналогичным образом организованная видеопамять. Адаптер, работающий в текстовом режиме, использует дополнительный блок — знакогенератор, который хранит растровые образы всех отображаемых символов. Эти образы представляются в виде битовых матриц размерностью от 8 × 8 до 9 × 16 бит, соответствующими точкам экрана знакоместа.

Текстовый адаптер также имеет аппаратные средства управления курсором: знакоместо, на которое указывают регистры координат курсора, оформляется особым образом. Обычно его выделяют мигающей полоской, размер и положение которой относительно знакоместа программируется. Заметим, что к такому выделению байт атрибутов символа не имеет отношения.

Стандартным текстовым режимом является формат 25 строк по 80 символов, хотя применяются и другие режимы. Требуемый объем видеопамяти по современным меркам ничтожный — всего 2 Кбайт для символов и 2 Кбайт для их атрибутов. При этом символы могут иметь хорошо читаемую матрицу разложения 9 × 14 и по 8 бит для атрибутов, определяющих цветовое оформление знакоместа. Частота считывания видеопамяти для регенерации изображения также невысока: за время прямого хода по строке должно быть считано всего 80 слов.

Поскольку в текстовом режиме в адаптер передаются только коды символов, заполнение всего экрана занимает в сотни раз меньше времени, чем для построения того же изображения в графическом режиме. Программный код вывода символов в текстовом режиме проще и компактнее, чем при программном формировании его растрового изображения. По этим причинам все видеоадаптеры имеют знакогенератор, дающий возможность работы и в текстовом режиме, а при переходе в графический режим знакогенератор отключается.

Современные адаптеры позволяют выводить символы (формировать их растровое изображение с заданным форматом знакоместа) и в графическом режиме.

При этом адаптер получает только команду с указанием координат отображаемых символов и сам поток кодов символов, после чего быстро строит их изображение, используя определенный шрифт.

Трехмерная графика.

Рассмотрим кратко методику построения трехмерных изображений и последовательность операций в алгоритме обработки трехмерной сцены.

Для того чтобы представить трехмерную сцену в цифровом виде, все объекты сцены преобразуются в **каркасные модели**, положение и форма которых полностью описываются координатами узлов (вершин) каркаса. Для описания поверхности модели используются так называемые **текстуры** — обычные двумерные прямоугольные изображения. Каркас модели «*обтягивается*» текстурой — для этого из каждого прямоугольного изображения необходимо «вырезать» многоугольник, соответствующий одной из граней каркаса.

Очевидно, что точность проработки модели достигается путем увеличения числа узлов каркаса и применения текстур высокого разрешения. Для этого используются наиболее простые из возможных многоугольников — **треугольники**. Таким образом, информация о трехмерной сцене представляет собой массив координат вершин всех треугольников, из которых состоят объекты сцены, файлы изображений (текстуры), координаты и данные о типе источников освещения и координаты «**виртуальной камеры**» — точки, откуда рассматривается трехмерная сцена.

Получение изображения трехмерной сцены на экране, т. е. создание двухмерного массива окрашенных экранных точек, состоит из трех этапов:

1) подготовка сцены;

2) геометрические преобразования и расчет освещенности — этап **T&L (Transform and Lighting)**;

3) растрирование и текстурирование.

Подготовка сцены — это собственно создание геометрических моделей объектов, размещение их в трехмерном виртуальном пространстве вместе с источниками освещения и виртуальной камерой и преобразование их к набору треугольников в случае, если модель была составлена из более сложных геометрических элементов. Для подготовки трехмерной сцены прикладная программа (например, игра или пакет трехмерного моделирования) использует ресурсы центрального процессора.

Геометрические преобразования заключаются в проецировании трехмерной сцены на плоскость экрана. Другими словами, координаты объектов сцены необходимо преобразовать в систему координат экрана. При этом необходимо отсечь все объекты (или части объектов), которые не попадают в поле зрения виртуальной камеры, и преобразовать размеры объектов в соответствии с их удаленностью от позиции виртуальной камеры для имитации перспективы.

Для привязки объекта к координатному пространству сцены используется так называемая *контрольная точка*, относительно которой с помощью афинных преобразований вычисляются координаты всех примитивов, составляющих объект. Для отсечения лишней информации определяется видимая область, заданная в виде усеченной пирамиды. Грани этой пирамиды служат секущими плоскостями, объекты и части объектов, оказавшиеся вне этой области отбрасываются, благодаря чему минимизируется количество обрабатываемых объектов. После этого координаты сцены приводятся к двумерным координатам экрана, а координата z — удаленность от плоскости экрана — заносится в так называемый z-буфер.

По завершении этапа геометрических преобразований трехмерная сцена оказывается представленной массивами двухмерных координат вершин в плоскости экрана, их z-координат и нормалей. Кроме того, для каждой вершины **рассчитывается специальный вектор освещенности**. Нормали вершин необходимы в том числе и для оптимизации путем отбрасывания заведомо невидимых треугольников — тех, нормаль которых направлена от экрана.

Данных, полученных после геометрических преобразований, недостаточно для того, чтобы получить двухмерное изображение. Переход от набора треугольников к набору точек называется **растрированием**: каждый треугольник сканируется и превращается в набор экранных точек, для каждой из которых вычисляется собственная z-координата.

Далее наступает этап **текстурирования**. Каждая экранная точка окрашивается — цвет выбирается из цветовой карты (одной или нескольких) в соответствии с координатами точки, уровнем прозрачности и т. п. Экранные точки — это, вообще говоря, прямоугольники конечных размеров, и при попадании на границу треугольника лишь часть их принадлежит треугольнику. Однако при растрировании экранная точка присваивается треугольнику полностью и в процессе текстурирования ее цвет рассчитывается исходя из этого приближения. В результате края объектов выглядят «зубчатыми», неестественными. Бороться с этим призваны *технологии сглаживания*, заметно усложняющие алгоритм текстурирования.

Еще одна проблема — необходимость масштабирования текстур для удаленных от плоскости экрана объектов трехмерной сцены. Очевидно, что при автоматическом масштабировании значительно снижаются четкость и реалистичность текстуры, что особенно заметно на изображениях с периодической структурой. Поэтому обычно используется не одна текстура, а набор из нескольких, предназначенных для объектов, которые расположены на разном расстоянии. Однако в этом случае возникает проблема «сшивки» текстур — граница перехода не должна быть заметна, а значит, вновь в дело должны вступить технологии сглаживания. Окончательное окрашивание точки экрана в нужный цвет называется *рендерингом*. После этого данные записываются в кадровый буфер и с помощью RAMDAC выводятся на экран.

Эволюция видеоадаптеров в области поддержки 3D-графики происходила путем постепенной передачи функций по подготовке трехмерного изображения от центрального процессора к адаптеру. Адаптеры, способные обрабатывать трехмерные сцены, стали называть **трехмерными ускорителями (3D-ускорители)**, а позже — **графическими процессорами**.

3D-ускорители первого поколения освобождали ЦП от выполнения функций только последнего этапа подготовки изображения — наложения текстур, остальные этапы выполнялись на программном уровне. Однако одно это уже обеспечивало значительное ускорение обработки трехмерных сцен.

В дальнейшем процесс текстурирования был значительно усовершенствован. За счет использования нескольких конвейеров рендеринга с несколькими текстурными блоками в каждом пиковая скорость заполнения сцены пикселами была доведена до таких значений, превышать которые уже нет никакого смысла из-за нехватки пропускной способности видеопамяти.

Во втором поколении ускорителей — а именно с этого поколения их стало принято называть графическими процессорами — на аппаратном уровне выполнялось не только текстурирование, но и преобразование координат и расчет освещенности треугольников (T&L). В значительной мере сняв остроту вопроса ресурсоемкости визуализации трехмерных сцен, первые модификации графических процессоров практически не предоставили дополнительных средств для улучшения реалистичности изображений, так как обладали ограниченным фиксированным набором функций для реализации оптических эффектов.

Проблема заключалась в том, что разработчики не имели возможности влиять на способ обработки отдельных вершин — все вершины и пикселы обрабатывались по единым правилам, с учетом лишь ограниченного набора дополнительных свойств и параметров.

В следующих поколениях графических процессоров был реализован более гибкий подход к расчету параметров вершин и пикселов — *шейдеры*. **Шейдер** — это блок обработки вершин или пикселов (соответственно вершинный или пиксельный шейдер), которому для расчета передается стандартизованный комплект данных, описывающий свойства вершины или пикселя. Структуру данных шейдера задает архитектура графического процессора и используемый разработчиком программный интерфейс. Наиболее популярным подобным программным интерфейсом является **Microsoft DirectX**.

Шейдеры не только предоставляют возможность гибко управлять параметрами вершин и пикселов. Гораздо важнее, что, помимо регистров данных, вершинные шейдеры содержат дополнительный регистр микрокода, с помощью которого разработчик может запрограммировать правила обработки данных, содержащихся в регистрах шейдера. Это дает возможность реализовать новые оптические эффекты и добиться существенного улучшения реалистичности изображений, в частности объектов со сложной рельефной поверхностью (например, шерсти животных).

Заметим, что с ростом производительности и функциональности графических процессоров увеличиваются объем и быстродействие видеопамяти. Современные графические процессоры содержат память объемом в сотни мегабайт.

На современном рынке массовых графических процессоров наблюдается борьба двух основных конкурентов — компаний NVIDIA и ATI. Несколько особняком от них держатся фирмы: Matrox, специализирующаяся на высококачественной 2D-графике, и SGI, выпускающая дорогие профессиональные графические рабочие станции.

В заключение отметим особенность современных видеоадаптеров, связанную с повсеместным переходом от CRT- к ЖК-мониторам. Для соединения адаптера с CRT-монитором используется 9-контактный VGA-разъем, по которому передается аналоговый сигнал, управляющий лучом ЭЛТ. ЖК-мониторам для работы требуется цифровой сигнал, но на момент их массового появления на рынке все потребительские видеоадаптеры использовали выход VGA. Быстрое, но не очень качественное решение заключается в получении аналогового сигнала от адаптера и последующем преобразовании его в цифровой вид внутри ЖК-монитора с помощью АЦП (аналого-цифрового преобразователя). Очевидно, качество сигнала при этом теряется.

В 1998 г. компании Fujitsu, Compaq, HP, IBM, Intel, NEC и Silicon Image основали группу DDWG (Digital Display Work Group), которая в апреле 1999 г. анонсировала стандарт **DVI 1.0 (Digital Visual Interface** — «цифровой видеоинтерфейс»), и с тех пор он являлся единым стандартам для подключения цифровых мониторов.

Как уже отмечалось, видеоадаптер создает цветовую информацию для каждого пикселя. Этот поток RAMDAC преобразует в аналоговый сигнал VGA. Когда данные передаются по DVI, то RAMDAC не используется. Вместо него поток параллельных данных поступает на специальный **передатчик TMDS (Transition Minimized Differential Signaling)**, который преобразует его в последовательный сигнал. Когда в мониторе сигнал поступает на приемник, то его последовательный характер вновь преобразуется в параллельный. Преобразование в последовательный сигнал для передачи по кабелю необходимо, поскольку последовательная передача менее подвержена помехам, чем параллельная, особенно на больших расстояниях.

Таким образом, особенность адаптеров ЖК-дисплеев по сравнению с адаптерами для ЭЛТ состоит в использовании блока TMDS вместо RAMDAC.

4.5.4.7. Звуковые адаптеры

Как известно, вся обрабатываемая информация, в том числе и аудиосигналы, представляется в ЭВМ в виде чисел. Реальный аудиосигнал — это сложное по форме колебание, некая сложная зависимость амплитуды звуковой волны от времени. Преобразование аналогового звукового сигнала в цифровой вид называется

аналогово-цифровым преобразованием, или **оцифровкой**. Процесс такого преобразования состоит в:

• замерах величины амплитуды аналогового сигнала с некоторым временным шагом — *дискретизации*;

• последующей записи полученных значений амплитуды в численном виде — *квантовании*.

Дискретизация по времени — это процесс получения мгновенных значений преобразуемого аналогового сигнала с определенным временным шагом, называемым **шагом дискретизации**. Количество осуществляемых в одну секунду замеров величины сигнала называют **частотой дискретизации**. Очевидно, что, чем выше частота дискретизации (т. е. тем чаще регистрируются значения амплитуды), тем более точное представление о сигнале мы получаем. Это рассуждение подтверждается **теоремой Котельникова**, согласно которой аналоговый сигнал с ограниченным спектром может быть точно описан дискретной последовательностью значений его амплитуды, если эти значения следуют с частотой, как минимум, вдвое превышающей наивысшую частоту спектра. Как известно, человеческое ухо способно различать звуковые сигналы частотой до 20 КГц, поэтому минимальная частота дискретизации при оцифровке звука должна составлять величину вдвое большую, т. е. 40 КГц.

Допустим, что для записи одного значения амплитуды сигнала в памяти ЭВМ отводится N бит. Соответственно с помощью одного N-битного слова можно описать 2^N разных положений. Представим теперь диапазон изменения амплитуды оцифровываемого звукового сигнала — **динамический диапазон сигнала** — в виде 2^{N-1} равных промежутков, разделив его на 2^N уровней — **квантов**. Теперь для записи каждого отдельного значения амплитуды его необходимо округлить до ближайшего уровня квантования. Этот процесс называется **квантованием по амплитуде**. Квантование по амплитуде — это процесс замены реальных (измеренных) значений амплитуды сигнала значениями, приближенными с некоторой точностью. Каждый из 2^N возможных уровней называется **уровнем квантования**, а расстояние между двумя ближайшими уровнями называется **шагом квантования**.

В случае линейного разбиения амплитудной шкалы на уровни квантование называют **линейным** (однородным). Очевидно, что точность округления зависит от выбранного количества (2^N) уровней квантования, которое, в свою очередь, зависит от количества бит (N), отведенных для записи значения амплитуды. Чем больше уровней квантования и чем ближе они друг к другу (для некоторого фиксированного диапазона изменения амплитуды расстояние между уровнями квантования обратно пропорционально их количеству), тем на меньшую величину приходится округлять измеренные значения амплитуды и, таким образом, тем меньше погрешность квантования. Число N называют **разрядностью квантова-**

ния. Считается, что погрешность квантования с разрядностью 16 бит остается для слушателя почти незаметной.

Описанный способ оцифровки сигнала — дискретизация сигнала во времени в совокупности с методом однородного квантования — называется *импульсно-кодовой модуляцией*, *ИКМ* (Pulse Code Modulation — PCM). Стандартный звуковой компакт-диск (CD-DA) хранит информацию в формате PCM с частотой дискретизации 44,1 КГц и разрядностью квантования 16 бит.

Для воспроизведения цифрового звука в ЭВМ применяются специальные устройства — **звуковые адаптеры** (контроллеры). Заметим, что не совсем корректно причислять звуковые адаптеры к устройствам вывода информации, так как все они могут выполнять и обратную операцию — ввод аналогового звукового сигнала в ЭВМ с преобразованием его в цифровую форму.

При воспроизведении звука в ЭВМ все начинается с потока цифровых аудиоданных, поступающих в звуковой контроллер. Для простоты будем считать, что поток состоит из несжатой звуковой информации в формате PCM, 44 КГц, 16 бит. Прикладная программа считывает информацию из файла на диске, преобразует в поток и доставляет его драйверу звукового адаптера, который, в свою очередь, направляет его в **цифровой сигнальный процессор** (**DSP**, **Digital Signal Processor**), или **звуковой процессор**.

Звуковой процессор определяет, требуется ли для данного звукового потока какая-либо предварительная обработка: возможно, его придется конвертировать в реальном масштабе времени в единственный воспринимаемый **цифро-аналоговым преобразователем** (**ЦАП**) формат. После этого DSP формирует пакеты и передает их в ЦАП. ЦАП принимает данные и формирует из них непрерывный аналоговый поток, иногда направляемый в микросхему аналогового предварительного усилителя, а иногда выводимый прямо на аналоговые разъемы адаптера.

При записи аналоговый звук поступает сперва на разъемы звукового адаптера, затем на **аналогово-цифровой преобразователь** (**АЦП**) — либо напрямую, либо через буфер в виде операционного усилителя. АЦП формирует пакеты, которые буферизуются и под контролем звукового процессора передаются в виде потока цифровых аудиоданных в принимающую программу. При необходимости звуковой процессор применяет к потоку какие-либо алгоритмы обработки в реальном масштабе времени.

С самого начала возможности звуковых контроллеров делились на воспроизведение звука, записанного в цифровой форме, и синтез музыки, записанной в стандартном формате **MIDI** (**Musical Instruments Digital Interface** — «цифровой интерфейс музыкальных инструментов»). Интерфейс MIDI был разработан в 1983 г. альянсом компаний изготовителей музыкальных синтезаторов (Roland, Korg и Yamaha). Основной целью его создания была стандартизация интерфейса управления клавишными синтезаторами, которые, оставаясь аналоговыми, имели

цифровое управление, что позволяло точнее корректировать параметры звука, а также сохранять в памяти все произведенные настройки.

Следующий шаг, сделанный компанией Roland, — создание **секвенсора** (**sequencer** — устройство, способное записывать, сохранять, редактировать и впоследствии воспроизводить последовательности управляющих команд для синтезаторов) на базе в то время уже широко распространенного IBM PC. Звуковая плата, созданная компанией Roland для ПК, называлась Roland **MPU 401** (**Music Processing Unit** — «блок обработки музыки») и получилась столь удачной, что ее интерфейс стал стандартом де-факто для работы с MIDI-устройствами и до сих пор остается актуальным. Любой современный звуковой контроллер обязательно имеет режим эмуляции MPU 401 на аппаратном или программном уровне.

Несинтезированный цифровой звук пришел с компанией Creative и их первой звуковой платой **Creative Sound Blaster**. Ее популярность была настолько велика, что название Sound Blaster стало нарицательным практически для всех звуковых плат. С тех пор звуковые адаптеры прочно заняли потребительский рынок, совершенствуясь с годами.

В 1997 г. был принят стандарт AC-97 (Audio Codec-97), определяющий архитектуру встроенных в материнскую плату звуковых контроллеров.

Современные звуковые адаптеры выполняются как в виде стандартных PCI-плат, так и в виде USB-устройств. Практически все они поддерживают формат DVD-Audio, обеспечивая частоту дискретизации 192 КГц с разрядностью квантования в 24 бита. При этом аналоговый сигнал может выводиться на акустическую систему домашнего кинотеатра с многоканальным звуком (поддерживаются форматы 5.1 и 7.1). Лидерами массового рынка являются компании Creative (внешние адаптеры) и Intel (интегрированные решения).

Вопросы для самоконтроля

1. Каковы структура и функции модуля ввода-вывода?
2. Что такое адресное пространство ввода-вывода и каких видов оно бывает?
3. Какие существуют методы управления вводом-выводом, и чем они отличаются друг от друга?
4. Поясните принцип работы контроллера прямого доступа к памяти.
5. В чем состоит канальная организация системы ввода-вывода?
6. В чем состоит шинная организация системы ввода-вывода?
7. Какие Вы знаете алгоритмы арбитража шин?
8. Какие основные тенденции в развитии шинных структур?
9. Что такое шины «большого» и «малого» интерфейса?
10. Охарактеризуйте известные Вам стандарты шин.
11. Из каких этапов состоит информационный доступ к жесткому диску?
12. Каковы конструкция и устройство НЖМД?
13. Какие виды НГМД Вы знаете?

14. В чем заключается идея дисковых массивов RAID?
15. Сравните между собой различные уровни RAID.
16. Какой принцип лежит в основе оптической дисковой памяти?
17. Охарактеризуйте основные классы оптических дисковых накопителей.
18. В чем состоит принцип действия флэш-памяти?
19. Какие Вы знаете устройства резервного копирования информации?
20. В чем разница между стекером и автозагрузчиком?
21. Какие виды переключателей используются в клавиатурах?
22. Чем различаются джойстик, «мышь» и световое перо?
23. Что такое ПЗС-датчик, где и как он используется?
24. Какой тип сканера подходит для быстрой оцифровки многостраничных документов?
25. Какой метод цветоделения в сканерах обеспечивает наивысшее качество и почему?
26. Охарактеризуйте область применения трехмерных сканеров.
27. В чем состоит принцип действия матричных принтеров? Каких видов они бывают?
28. Перечислите преимущества и недостатки струйной печати.
29. Поясните схему действия лазерного принтера.
30. Какие типы и классы мониторов Вы знаете?
31. Как устроена ЖК-панель?
32. Что такое яркость, контрастность и цветовая температура монитора?
33. Поясните разницу между временем отклика и временем послесвечения.
34. Сравните преимущества и недостатки ЭЛТ- и ЖК-мониторов.
35. Как осуществляется вывод изображения на экран в текстовом режиме?
36. Что такое RAMDAC, и как определить частоту обновления экрана?
37. Охарактеризуйте этапы формирования трехмерного изображения на экране.
38. Что такое шейдер?
39. Что такое частота дискретизации и разрядность квантования звукового сигнала?
40. Какие преобразования звукового сигнала осуществляются в звуковом адаптере?

ГЛАВА 5. ПАРАЛЛЕЛЬНЫЕ ЭВМ И ВС

5.1. ТИПЫ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ИНФОРМАЦИИ

Под термином «**параллельная обработка**» мы будем понимать одновременное выполнение заданий, шагов (пунктов) заданий, программ, подпрограмм, циклов, операторов и команд. Параллельная обработка информации может применяться с двумя основными целями:

1) повышение производительности ЭВМ и ВС не за счет совершенствования элементной базы, а за счет эффективной организации вычислительных процессов;

2) обеспечение высокой надежности ВС за счет дублирования вычислительной аппаратуры.

Повышение производительности ЭВМ и ВС — основная цель применения параллельной обработки, по этой причине параллельную архитектуру имеют такие ЭВМ, как многопроцессорные серверы, мэйнфреймы и супер-ЭВМ.

Параллельная обработка информации может производиться на нескольких уровнях (рис. 5.1).

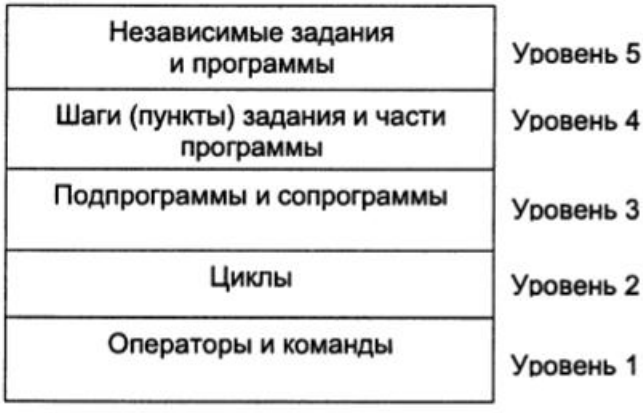


Рис. 5.1. Уровни параллелизма

Очевидно, что чем ниже уровень, тем мельче дробление программных процессов, тем мельче, как принято говорить, «**зерно параллелизма**». В общем случае возможно реализовать параллелизм как на отдельном уровне, так и на нескольких одновременно. Независимая однопроцессорная обработка реализует параллелизм на уровне 1. Векторная обработка заключается в параллельном выполнении циклов на уровне 2 и может производиться как на одном, так и нескольких процессорах. Уровни 3 и 4 соответствуют многопроцессорным ВС. Параллелизм уровня 5 характерен для многомашинных вычислительных комплексов.

Существуют два основных способа организации параллельной обработки:

• совмещение во времени этапов решения разных задач;

• одновременное решение различных задач или частей одной задачи.

Первый путь — совмещение во времени этапов решения разных задач — это **мультипрограммная обработка** информации. Мультипрограммная обработка уже давно и широко применяется для повышения производительности ЭВМ и ВС. Подробное рассмотрение мультипрограммной обработки относится к теме «Операционные системы» и выходит за рамки настоящего учебника.

Второй путь — одновременное решение различных задач или частей одной задачи — возможен только при наличии нескольких обрабатывающих устройств. При этом используются те или иные особенности задач или потоков задач, что позволяет осуществить распараллеливание.

Можно выделить следующие типы параллелизма, позволяющие реализовать алгоритмические особенности отдельных задач и их потоков:

1) естественный параллелизм независимых задач;

2) параллелизм объектов или данных;

3) параллелизм ветвей задачи или программы.

Рассмотрим эти типы параллелизма.

1. Естественный параллелизм независимых задач заключается в том, что на вход ВС поступает непрерывный поток не связанных между собой задач, т. е. решение любой задачи не зависит от результатов решения других задач. В этом случае использование нескольких обрабатывающих устройств при любом способе комплексирования (объединения в систему) повышает производительность системы.

Характерным примером естественного параллелизма является поступление пользовательских запросов на информационный web-сайт. Каждый запрос порождает отдельную процедуру его исполнения, которая не зависит от других подобных процедур.

2. Параллелизм объектов или данных имеет место тогда, когда по одной и той же (или почти по одной и той же) программе должна обрабатываться некоторая совокупность данных, поступающих в систему одновременно.

Это могут быть, например, задачи обработки сигналов от радиолокационной станции: все сигналы обрабатываются по одной и той же программе. Другой пример — обработка информации от датчиков, измеряющих одновременно один и тот же параметр и установленных на нескольких однотипных объектах.

Программы подобного типа могут быть различного объема и сложности, начиная от очень простых, содержащих несколько операций, до больших программ в сотни и тысячи операций. При этом параллельность выполнения операций достигается путем увеличения числа обрабатывающих устройств, каждое из которых способно автономно выполнять последовательность команд над отдельной совокупностью данных. Часто основной особенностью таких программ (в частности программ обработки векторов и матриц) является то, что одна и та же команда

должна выполняться над большой совокупностью элементарных, связанных между собой некоторым образом данных, и соответствующую операцию можно производить над всеми данными одновременно. При этом время решения задачи сокращается пропорционально числу обрабатывающих устройств.

3. Параллелизм ветвей задачи или программы — один из наиболее распространенных типов параллелизма в обработке информации. Он заключается в том, что при решении одной задачи могут быть выделены отдельные ее части — ветви, которые при наличии нескольких обрабатывающих устройств могут выполняться параллельно. При этом одновременно могут обрабатываться только **независимые ветви** задачи, т. е. такие ее части, для которых соблюдаются следующие условия:

- ни одна из выходных величин этих ветвей задачи не является входной величиной для другой такой ветви (отсутствие функциональных связей);
- условия выполнения одной ветви не зависят от результатов или признаков, полученных при выполнении других ветвей (независимость по управлению).

Хорошее представление о параллелизме ветвей дает ярусно-параллельная форма (ЯПФ) программы, пример которой приведен на рис. 5.2.

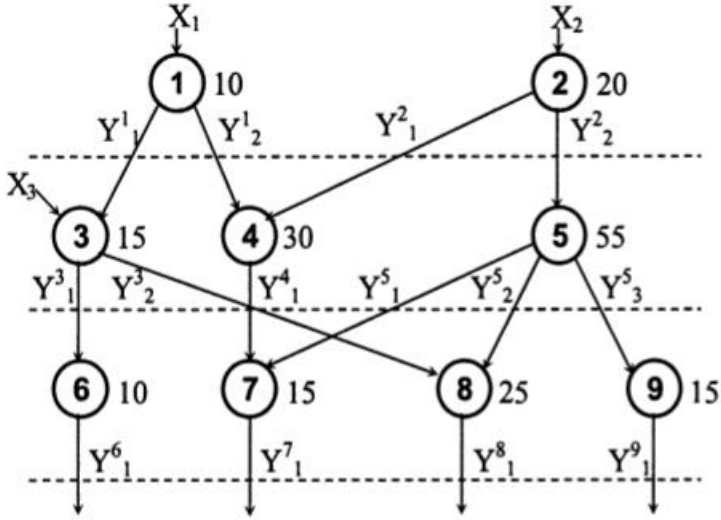


Рис. 5.2. Пример ярусно-параллельной формы программы

Программа представлена в виде совокупности ветвей, расположенных на нескольких уровнях — ярусах. Кружками с цифрами внутри обозначены ветви. Длина ветви представляется цифрой, стоящей около кружка и означающей, сколько временных единиц выполняется данная ветвь. Стрелками показаны входные данные и результаты обработки. Входные данные обозначаются символом *X*, выходные данные — символом *Y*. Символы *X* имеют нижние цифровые индексы, обозначающие номера входных величин; символы *Y* имеют цифровые индексы и внизу, и вверху; цифра вверху соответствует номеру ветви, при выполнении ко-

торой получен данный результат, а цифра внизу означает порядковый номер результата, полученного при реализации данной ветви программы. На одном ярусе размещаются независимые ветви задачи, не связанные друг с другом, т. е. результаты решения какой-либо ветви данного яруса не являются входными данными для другой ветви этого же яруса.

Изображенная на рис. 5.2 программа содержит 9 ветвей, расположенных на 3 ярусах. На примере этой, в общем, достаточно простой программы, можно выявить преимущества вычислительной системы, включающей несколько обрабатывающих устройств, и проблемы, которые при этом возникают.

Примем, что длина i-й ветви представляется числом временных единиц t_i, которые требуются для ее исполнения. Тогда нетрудно подсчитать, что для исполнения всей программы на 1 процессоре потребуется время T_1:

$$T_1 = \sum_{i=1}^{9}(10+20+15+30+55+10+15+25+15) = 195.$$

Если представить, что программа выполняется двумя обрабатывающими устройствами (процессорами), работающими независимо друг от друга, то время решения задачи сократится. Однако это время, как нетрудно видеть, будет различным в зависимости от последовательности выполнения независимых ветвей.

Рассмотрим, например, такой вариант выполнения программы, представленной на рис. 5.2: пусть процессор 1 выполняет ветви 1-3-4-6-7-9, а процессор 2 выполняет ветви 2-5-8. На рис. 5.3 представлены временные диаграммы выполнения процессорами ветвей программы.

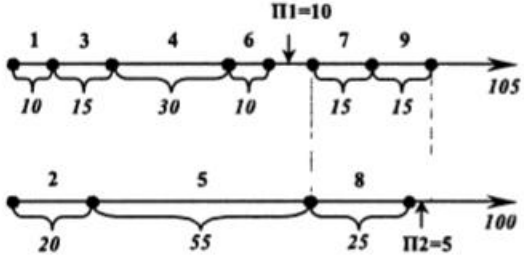


Рис. 5.3. Разложение ветвей программы по двум процессорам

Нетрудно подсчитать, что процессор 1 затрачивает 105, а процессор 2 — 100 единиц времени. При этом имеются два промежутка времени, когда один из процессоров вынужденно простаивает — П1 длительностью 10 единиц и П2 длительностью 5 единиц времени. Промежуток П1, во время которого работает только процессор 2, образовался из-за того, что ветвь 7 зависит от ветви 5 (к моменту завершения ветви 6 еще не готовы данные Y_1^5). Промежуток П1, во время которого работает только процессор 1, образовался по причине окончания счета процессором 2.

Таким образом, на системе из двух процессоров наша программа будет выполнена полностью не менее, чем за 105 единиц времени. Величину, характеризующую уменьшение времени решения задачи на нескольких процессорах по сравнению с использованием одного процессора, называют **ускорением счета** S и рассчитывают как

$$S = \frac{T_1}{T_P}, \tag{5.1}$$

где T_p — время решения задачи на p процессорах. Кроме ускорения, используют такую величину, как **коэффициент распараллеливания** $K_п$, который рассчитывается как

$$K_{\text{П}} = \frac{T_1}{P \cdot T_P}. \tag{5.2}$$

Коэффициент распараллеливания изменяется от 0 до 1 (от 0 до 100%) и отражает эффективность использования вычислительных ресурсов. В нашем примере нетрудно посчитать, что ускорение $S = 195/105 = 1{,}86$, а коэффициент распараллеливания $K_п = 0{,}93$. Как видим, по причине простоев одного из процессоров ускорение счета значительно меньше двух, т. е. количества используемых процессоров. Заметим, что в нашем примере не учитывались временные задержки, связанные с переключением контекстов программы (смены ветвей) и передачи данных от одной ветви к другой. Тем не менее в силу алгоритмических особенностей программы часть вычислений в промежутки П1 и П2 производится только одним процессором, т. е. фактически последовательно.

Рассмотрим обобщенный случай программы, в которой алгоритмически доля последовательных вычислений (отношение времени последовательных вычислений к общему времени счета программы) составляет некоторую величину f. В этом случае время выполнения программы на системе из p процессоров не может быть меньше величины

$$T_P \geq \frac{T_1 - T_1 \cdot f}{p} + T_1 \cdot f. \tag{5.3}$$

Первое слагаемое определяет время выполнения распараллеленной части программы, а второе слагаемое — время выполнения последовательной части программы. Нетрудно видеть, что ускорение счета в этом случае не будет превышать величины

$$S \leq \frac{1}{f + \frac{1-f}{p}}. \tag{5.4}$$

Данное соотношение носит название **закона Амдала**. На примере программы рис. 5.2 можем видеть, что доля последовательных вычислений составляет $f = 15/195$. Подставляя эту величину в формулу закона Амдала, получаем для системы из двух процессоров максимальное ускорение 1,86 раза, что соответствует ранее рассчитанному значению.

Для иллюстрации действия закона Амдала приведем следующий пример. Пусть доля последовательных вычислений в некоторой программе составляет 10%. Тогда максимальное ускорение счета на 100 процессорах не превысит 9,2. Коэффициент распараллеливания составит всего лишь 9,2%. На 10 процессорах ускорение составит 5,3, а коэффициент распараллеливания — 53%. Нетрудно видеть, что даже такая небольшая доля последовательных вычислений уже на теоретическом уровне, без учета неизбежных задержек в реальной ВС, серьезно ограничивает возможности масштабирования программы.

Определим, какая должна быть максимальная доля f последовательных вычислений в программе, чтобы было возможно получить наперед заданное ускорение счета S с максимальным коэффициентом распараллеливания $K_п$. Для этого выразим из закона Амдала долю последовательных вычислений:

$$f \le \frac{p - S}{S \cdot (p - 1)}. \tag{5.5}$$

Из соотношения (5.5) видно, что для того, чтобы допустить наличие последовательных вычислений в программе (f должно быть больше 0), число процессоров p должно быть больше предполагаемого ускорения S. При этом условии максимальный коэффициент распараллеливания $K_п$ получается при числе процессоров $p = S + 1$. Таким образом, доля последовательных вычислений должна составлять не более

$$f \le \frac{1}{S^2}. \tag{5.6}$$

Соотношение (5.6) определяет очень важное **следствие** из закона Амдала. *Для того чтобы ускорить программу в q раз, необходимо ускорить не менее, чем в q раз, не менее, чем* $(1 - \frac{1}{q^2})$*-ю часть программы*. Например, чтобы получить ускорение в 100 раз, необходимо распараллелить 99,99% всей программы.

Кроме алгоритмического распараллеливания, для того чтобы с помощью нескольких обрабатывающих устройств решить задачу, имеющую параллельные ветви, необходима соответствующая организация процесса, которая определяет пути решения задачи и вырабатывает необходимую информацию о готовности каждой ветви. Однако все это относительно легко реализовать тогда, когда известна достаточно точно длительность выполнения каждой ветви. На практике

это бывает крайне редко: в лучшем случае имеется та или иная временная оценка. Поэтому организация оптимального или близкого к оптимальному графика работы является достаточно сложной задачей.

Следует отметить также и определенные сложности, связанные с выделением независимых ветвей при разработке программ. Вместе с тем при решении многих сложных задач только программирование с выделением независимых ветвей позволяет существенно сократить время решения. В частности, хорошо поддаются параллельной обработке такого типа задачи матричной алгебры, линейного программирования, спектральной обработки сигналов, прямые и обратные преобразования Фурье и др.

5.2. КЛАССИФИКАЦИЯ ПАРАЛЛЕЛЬНЫХ ВС

5.2.1. Признаки классификации параллельных ВС

В литературе описаны десятки систем классификации параллельных ВС, которые различаются по количеству и типам учитываемых признаков и по глубине иерархии классификации. Все эти системы классификации можно с определенной степенью условности разделить на четыре основные группы по

1) системно-функциональным характеристикам ВС и характеристикам управления;

2) характеристикам топологии размещения процессоров и машин в ВС и связей между их элементами;

3) характеристикам собственно способов параллельной обработки;

4) структурно-функциональным характеристикам степени и типа параллельной обработки информации.

Основные признаки систем классификации **первой группы** характеризуют *однородность* и *неоднородность* ВС, *централизацию* и *децентрализацию* управления и его *иерархию*, *универсальность* и *специализацию*, *степень и уровень связанности* процессоров и машин, *степень закрепленности функций* за процессорами и машинами, режимы работы и т. п.

Системы **второй группы** содержат признаки, отражающие *конструктивную разделимость* или *неразделимость* ВС, *конфигурацию* соединительных шин и коммутаторов, *изменяемость* или *неизменяемость* структуры и др. Различают следующие основные топологии размещения элементов параллельной ВС: линейка, кольцо, дерево, решетка, звезда, гиперкуб, тор. Подробнее данные топологии будут рассмотрены в разделе «Распределенные составные коммутаторы». Соединительные шины и коммутаторы также могут иметь различную конфигурацию, например коммутаторы Клоза, баньян-сети (дельта- и омега-сети) и др.

Признаки **третьей группы** характеризуют способы параллельной обработки информации, такие, как *многомашинный*, *многопроцессорный*, *магистральный*, *векторный*, *матричный* и *ассоциативный*.

Признаки **четвертой группы** определяют *единственные* или *множественные потоки команд* и *данных*, их *структуру* и другие характеристики.

Для сравнительной характеристики архитектур параллельных ВС в настоящее время чаще других применяется последняя группа классификаций, которая в основном опирается на классификацию М. Дж. Флинна.

5.2.2. Классификация Флинна

Вычислительный процесс в ВС можно представить как *воздействие определенной последовательности команд программы (**поток команд**) на соответствующую последовательность данных (**поток данных**)*, обрабатываемых этой последовательностью команд. Следовательно, различные способы организации любого типа обработки информации можно рассматривать как способы организации одновременного воздействия одного или нескольких потоков команд на один или несколько потоков данных.

Для такой классификации в 1966 г. М. Дж. Флинном было введено понятие **множественности** потоков команд и данных. Следуя ему, под **множественным потоком команд (данных)** мы будем понимать наличие в системе нескольких последовательностей команд, находящихся в стадии реализации (нескольких последовательностей данных, подвергающихся обработке командами).

Рассматривая ВС с точки зрения единственности и множественности потоков команд и данных, их можно разбить на четыре основных класса.

1. **Системы с одиночным потоком команд и одиночным потоком данных (ОКОД)**, Single Instruction stream / Single Data stream (**SISD**).

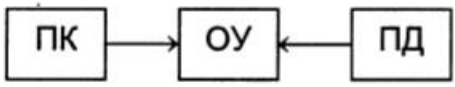


Рис. 5.4. Класс SISD (ОКОД)

Одиночный поток команд (ПК) поступает на вход единственного обрабатывающего устройства (ОУ), которое обрабатывает одиночный поток данных (ПД). Традиционно к классу SISD относят все возможные ЭВМ последовательного типа. В таких машинах есть только один поток команд, все команды обрабатываются последовательно друг за другом и каждая команда инициирует одну операцию с одним потоком данных. Как правило, не учитывается тот факт, что для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка.

2. **Системы с множественным потоком команд и одиночным потоком данных (МКОД)**, Multiple Instruction stream / Single Data stream (**MISD**).

Система класса MISD должна иметь несколько обрабатывающих устройств, на каждое из которых поступает свой поток команд. Обработке при этом подвергается единственный поток данных, который без ограничения общности может

быть разбит на несколько последовательно обрабатываемых каждым ОУ порций. Очевидное предназначение MISD-систем — конвейерная обработка информации, поэтому часть исследователей относят к данному классу конвейерные машины. Однако значительная часть ученых (среди них был и Флинн) склонна считать данный класс пустым. В настоящий момент в научной среде нет однозначного толкования данного вопроса. В дальнейшем, рассматривая конвейерную обработку информации, мы будем иметь в виду класс MISD.

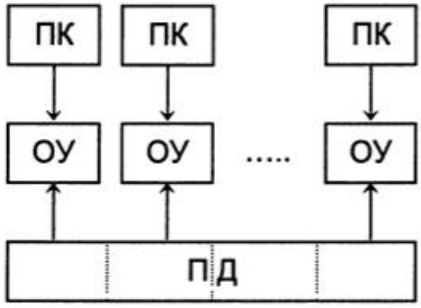


Рис. 5.5. Класс MISD (МКОД)

3. **Системы с одиночным потоком команд и множественным потоком данных (ОКМД)**, Single Instruction stream / Multiple Data stream (**SIMD**).

Единственный поток команд поступает одновременно на вход нескольких обрабатывающих устройств, каждое из которых выполняет общую для всех ОУ команду над очередным элементом своего потока данных. В класс SIMD входят векторные, матричные и ассоциативные ВС, в которых одна команда может выполняться над множеством элементов данных.

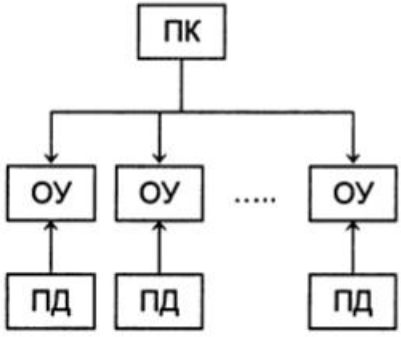


Рис. 5.6. Класс SIMD (ОКМД)

4. **Системы с множественным потоком команд и множественным потоком данных (МКМД)**, Multiple Instruction stream / Multiple Data stream (**MIMD**).

В MIMD-системе имеется множество, как правило, связанных между собой, обрабатывающих устройств, каждое из которых управляется собственным потоком

команд и обрабатывает собственный поток данных. В данный класс попадают практически все многопроцессорные и многомашинные ВС.

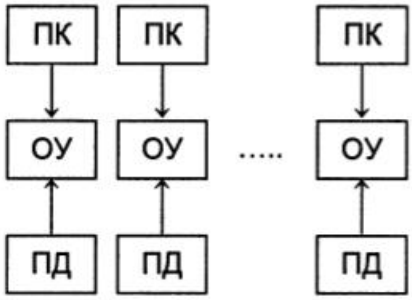


Рис. 5.7. Класс MIMD (МКМД)

Как нетрудно заметить, классификация Флинна имеет существенные недостатки: во-первых, неопределен состав класса MISD, во-вторых, перегружен класс MIMD, который включает в себя множество ВС с довольно различной архитектурой, в-третьих, в классификацию Флинна однозначно не укладываются так называемые *нетрадиционные параллельные архитектуры* — гибридные системы, ВС, управляемые потоком данных и др. Но, несмотря на эти недостатки, классификация Флинна является наиболее употребительной вплоть до сегодняшнего дня.

В 1984 г. К. Ванг и Ф. Бриггс сделали некоторые дополнения к классификации Флинна. Оставляя четыре ранее введенных базовых класса (SISD, SIMD, MISD, MIMD), авторы внесли следующие изменения.

Класс SISD разбивается на два подкласса:

- системы с **единственным** функциональным устройством;
- системы, имеющие в своем составе **несколько** функциональных устройств.

В класс SIMD также вводятся два подкласса:

- системы с **пословно-последовательной** обработкой информации;
- системы с **разрядно-последовательной** обработкой.

В классе MIMD авторы различают

- вычислительные системы со **слабой связью** между процессорами, к которым относятся все системы с *распределенной памятью*;
- вычислительные системы с **сильной связью** (системы с *общей памятью*).

5.2.3. Классификация Фенга

Помимо М. Флинна, попытки классификации параллельных ВС предпринимали и другие ученые. В 1972 г. Т. Фенг предложил достаточно простую классификацию, в основе которой лежат две характеристики — число бит n в машинном слове (фактически длина машинного слова), обрабатываемых параллельно при выполнении машинных команд, и число слов m, обрабатываемых одновременно данной вычислительной системой. По Фенгу, функционирование любой ЭВМ или

ВС можно представить как параллельную обработку n битовых слоев, на каждом из которых независимо преобразуются m бит. Опираясь на такую интерпретацию, вторую характеристику обычно называют *шириной битового слоя*.

Каждая вычислительная система C может быть описана парой чисел (n, m) и представлена точкой на плоскости в системе координат «длина слова — ширина битового слоя». Площадь прямоугольника со сторонами n и m определяет интегральную характеристику потенциала параллельности P архитектуры и носит название **максимальной степени параллелизма** вычислительной системы:

$$P(C) = m \cdot n. \tag{5.7}$$

В качестве примера рассмотрим вычислительный кластер из 4 вычислительных модулей, построенных на платформе процессора Intel Xeon HT 3,2 МГц. Каждый процессор имеет длину машинного слова 32 бита и по два набора конвейерных функциональных устройств, которые, с учетом технологии Hyper Threading, способны одновременно обрабатывать 2 потока вычислений. При этом конвейер функциональных устройств имеет 20 ступеней. Таким образом, в кластере одновременно могут обрабатываться $(4_{\text{процессора}} \times 2_{\text{потока}} \times 20_{\text{ступеней}}) = 160$ машинных слов. Указанная вычислительная система по Фенгу может быть описана парой (32, 160), причем максимальная степень параллелизма составит $32 \times 160 = 5120$.

На основе введенных понятий все вычислительные системы в зависимости от способа обработки информации, заложенного в их архитектуру, можно разделить на четыре класса.

- **Разрядно-последовательные, пословно-последовательные ($n = m = 1$).** В каждый момент времени такие ВС обрабатывают только один двоичный разряд. Представителем данного класса служит давняя система MINIMA с естественным описанием (1,1).
- **Разрядно-параллельные, пословно-последовательные ($n > 1, m = 1$).** К данному классу принадлежат большинство классических ЭВМ последовательного типа (если не учитывать конвейерность функциональных устройств современных микропроцессоров), например: PDP-11 с описанием (16,1), IBM S/360 и VAX 11/780 — обе с описанием (32,1).
- **Разрядно-последовательные, пословно-параллельные ($n = 1, m > 1$).** Вычислительные системы данного класса состоят из большого числа одноразрядных процессорных элементов, каждый из которых может независимо от остальных обрабатывать свои данные. Типичными примерами служат ассоциативные системы, такие, как STARAN (1, 256).
- **Разрядно-параллельные, пословно-параллельные ($n > 1, m > 1$).** Большая часть существующих параллельных вычислительных систем, обрабатывая одновременно $m \cdot n$ двоичных разрядов, принадлежит именно к этому классу.

Недостатки предложенной классификации достаточно очевидны и связаны со способом вычисления ширины битового слоя m. По существу Фенг не делает

никакого различия между процессорными матрицами, векторно-конвейерными и многопроцессорными системами. Не делается акцент на том, за счет чего ЭВМ или ВС может одновременно обрабатывать более одного слова: множественности функциональных устройств, их конвейерности или же какого-то числа независимых процессоров. Если в системе N независимых процессоров имеют каждый по F конвейерных функциональных устройств с длиной конвейера L, то для вычисления ширины битового слоя надо просто найти произведение данных характеристик.

Тем не менее классификация Фенга представляет значительную ценность за счет **введения единой числовой метрики** для всех типов ЭВМ и ВС, которая вместе с описанием потенциала вычислительных возможностей конкретной архитектуры позволяет сравнить любые две системы между собой.

5.2.4. Классификация Дункана. Понятие о нетрадиционных параллельных архитектурах

В 1990 г. Р. Дункан предложил собственную классификацию архитектур параллельных вычислительных систем, позволяющую в некоторой степени преодолеть основные недостатки классификаций Флинна и Фенга.

По мнению Дункана, классификация должна отвечать следующим требованиям.

1. Из класса параллельных машин должны быть исключены те, в которых параллелизм заложен лишь на уровне операторов и команд, включая:

- конвейеризацию на этапе подготовки и выполнения команды;
- наличие в процессоре нескольких функциональных устройств, работающих независимо, в частности, возможность параллельного выполнения логических и арифметических операций;
- наличие отдельных процессоров ввода-вывода, работающих независимо и параллельно с центральными процессорами.

Такой подход можно считать вполне правомерным, так как параллелизм уровня команд и операторов присутствует практически во всех современных ЭВМ и ВС, следовательно, все они могут быть классифицированны как «параллельные», что заведомо станет недостатком классификации.

2. Классификация должна быть согласованной с классификацией Флинна, показавшей правильность выбора идеи единственности и множественности потоков команд и данных.

3. Классификация должна описывать ВС, которые однозначно не укладываются в систематику Флинна, в том числе так называемые ВС с нетрадиционной архитектурой.

Учитывая вышеизложенные требования, Дункан дает следующее неформальное определение параллельной архитектуры.

Параллельная архитектура — это такой способ организации вычислительной системы, при котором допускается, чтобы множество процессоров (простых или сложных) могли бы работать одновременно, взаимодействуя по мере надобности друг с другом. Согласно данному определению, Дункан предлагает следующую систематизацию параллельных архитектур (рис. 5.8).

Рис. 5.8. Классификация Дункана

Три базовых класса строятся на очень простой основе: процессоры ВС работают либо синхронно, либо независимо друг от друга, либо в архитектуру системы заложена та или иная модификация идеи MIMD. На следующем уровне автор детализирует каждый из этих трех классов, включая ВС с нетрадиционной архитектурой, для каждой из которых ниже будет приведено краткое описание.

Систолические архитектуры (их чаще называют систолическими массивами) представляют собой множество процессоров, объединенных регулярным образом. Топология объединения может быть любой, хотя чаще всего используются «линейка» и «решетка». Обращение к памяти может осуществляться только через определенные процессоры на границе массива. Выборка операндов из памяти и передача данных по массиву и вычисления осуществляются в одном и том же темпе по тактам синхронизации. Каждый из процессоров массива «знает», от кого он должен принять входные данные и кому должен отдать результаты на каждом шаге счета. При этом каждый процессор на каждом шаге вычислений выполняет небольшую инвариантную последовательность действий.

Гибридные MIMD/SIMD архитектуры, dataflow, reduction и wavefront вычислительные системы осуществляют параллельную обработку информации на

основе асинхронного управления, как и MIMD-системы. Но они выделены в отдельную группу, поскольку все имеют ряд специфических особенностей, которыми не обладают системы, традиционно относящиеся к MIMD.

Архитектура MIMD/SIMD предполагает, что в MIMD-системе можно выделить группу процессоров, представляющую собой подсистему, работающую в режиме SIMD. Типичным представителем данного подкласса будут являться *векторно-параллельные* системы, такие, как NEC Earth Simulator.

Архитектура *dataflow* представляет ВС, *управляемые потоком данных*. В отличие от традиционной фон-неймановской архитектуры, где выборка команд на исполнение осуществляется по значению программного счетчика, в модели dataflow команда может выполняться сразу же, как только вычислены необходимые операнды. Таким образом, последовательность выполнения команд определяется зависимостью по данным, которая может быть выражена, например, в форме графа.

Модель вычислений, применяемая в *редукционных* (reduction) машинах иная и состоит в следующем: команда становится доступной для выполнения тогда и только тогда, когда результат ее работы требуется другой, доступной для выполнения, команде в качестве операнда.

Волновые массивы (wavefront array) объединяют в себе идею систолической обработки данных и модель вычислений, используемой в dataflow. В данной архитектуре процессоры объединяются в модули и фиксируются связи, по которым процессоры могут взаимодействовать друг с другом. Однако в противоположность ритмичной работе систолических массивов данная архитектура использует асинхронный механизм связи с подтверждением, из-за чего «фронт волны» вычислений может менять свою форму по мере прохождения по всему множеству процессоров.

5.2.5. Современная классификация параллельных ВС

Как уже отмечалось, начало в конце 80-х годов XX в. массового выпуска микропроцессоров дало толчок бурному развитию параллельных ВС, в том числе и в области построения новых архитектур. Естественно, классификация Дункана была не в состоянии предвидеть появление новых архитектур и их систематизировать.

Построим классификацию параллельных ВС с учетом вновь появившихся современных архитектур. В основу классификации (рис. 5.9) положим четыре базовых класса Флинна (**SISD**, **SIMD**, **MISD**, **MIMD**), которые разбиваются на подклассы в соответствии с дополнениями Ванга и Бриггса.

Класс SISD разделяется на системы с **одним** функциональным устройством (ФУ) и **несколькими** ФУ.

Класс SIMD образуют два подкласса — **разрядно-последовательных** и **пословно-последовательных** ВС.

В классе MIMD выделим **сильно-** и **слабосвязанные** ВС, а также ВС, **использующие идеи MIMD**, подклассы которых (MIMD/SIMD, dataflow, reduction, wavefront) образуются в соответствии с классификацией Дункана.

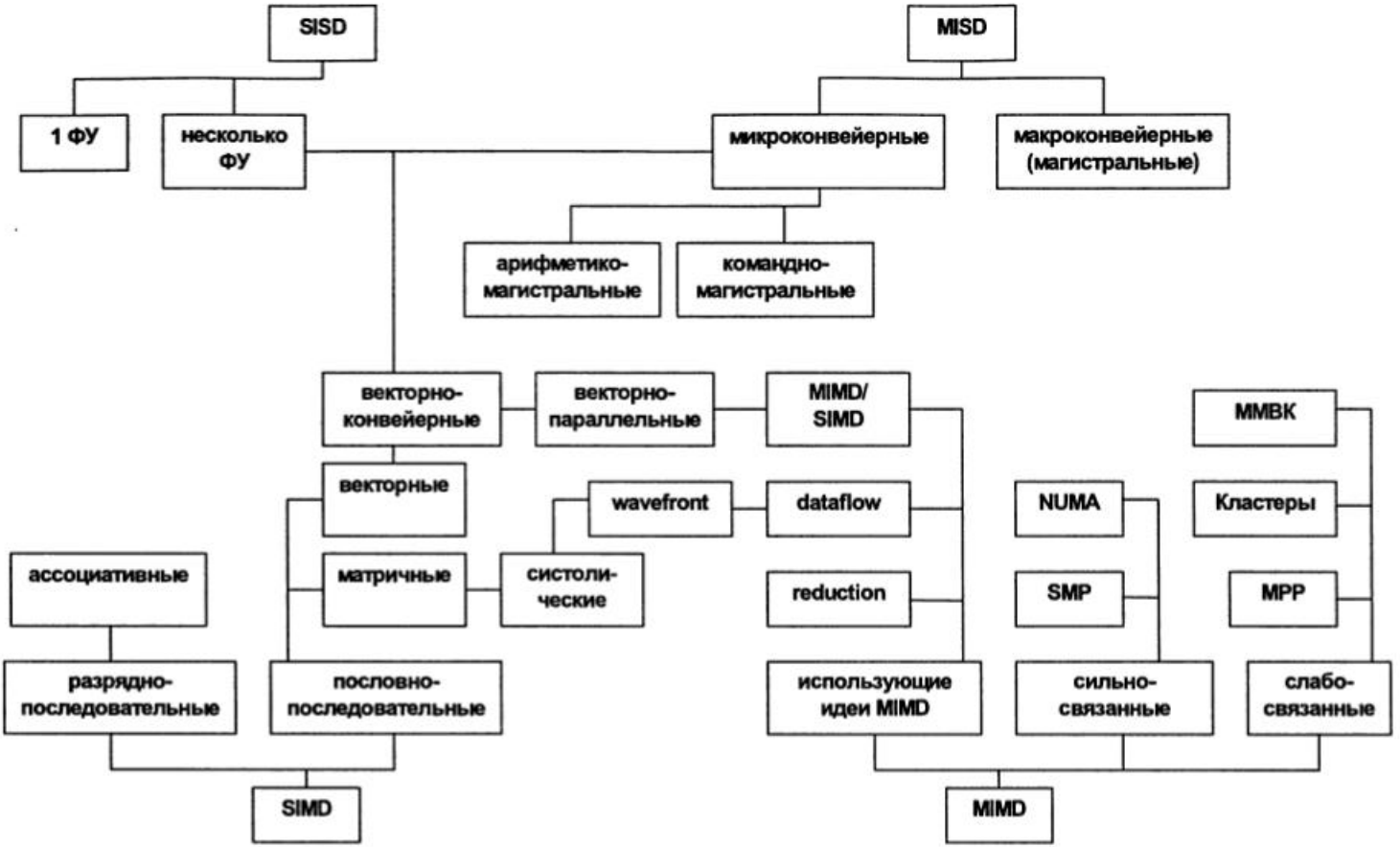


Рис. 5.9. Современная классификация параллельных ВС

Наконец к классу MISD отнесем ЭВМ и ВС, использующие идею **конвейерной обработки**. Параллелизм на уровне операторов и команд реализуется в **микроконвейерных системах**. При этом системы, способные разбивать на ступени непосредственное исполнение одной команды, образуют подкласс **арифметико-магистральных** систем, а системы, конвейеризующие все этапы выполнения команды (выборка из памяти, дешифрация, выборка операндов, исполнение, запись результатов), назовем **командно-магистральными**.

Кроме этого, выделен подкласс **макроконвейерных (магистральных)** ВС, способных многократно решать одну и ту же вычислительную задачу. При этом задача разбивается на последовательные части, каждая из которых выполняется на отдельном процессоре (обрабатывающем устройстве). Выходные данные одной части являются входными для следующей. За счет организации подобного макроконвейера при многократном повторении расчетов с разными входными данными получается существенный выигрыш в производительности.

Переходя к подклассам SIMD, видим, что к разрядно-последовательным SIMD-системам можно отнести **ассоциативные** ВС, а к пословно-последовательным — **векторные** и **матричные**. **Систолические массивы** можно рассматривать как отдельный подкласс матричных архитектур.

Среди сильносвязанных MIMD-архитектур выделяют **симметричные мультипроцессоры (SMP)**, имеющие сосредоточенную общую память и **архитектуры с неоднородным доступом к памяти (NUMA)**, в которых логическая общая память физически распределена по узлам системы.

Слабосвязанные MIMD-системы представлены **массово-параллельными (MPP)**, **кластерными** архитектурами, а также **многомашинными вычислительными комплексами (ММВК)**. MPP-системы представляют собой совокупность *специализированных вычислительных модулей*, объединенных *высокоскоростными межпроцессорными каналами связи*.

Кластерные системы также являют собой пример массового параллелизма, только их сборка осуществляется на основе *стандартных промышленных комплектующих*.

На схеме также показано, что ряд ВС могут быть одновременно отнесены к нескольким классам или подклассам. Сочетание принципов микроконвейерной и векторной обработки дает **векторно-конвейерные** архитектуры; привнося в них идеи MIMD/SIMD, получаем **векторно-параллельные (PVP)** ВС. Машины **wavefront** являются гибридом ВС, управляемых потоком данных (dataflow) и систолических массивов.

5.3. ИСТОРИЯ РАЗВИТИЯ ПАРАЛЛЕЛИЗМА В АРХИТЕКТУРЕ ЭВМ

Идеи параллелизма начали внедряться в архитектуру ЭВМ очень давно. Как и многие другие усовершенствования, те или иные элементы параллельной обработки сначала появлялись в супер-ЭВМ — в самых передовых, единичных компьютерах

своего времени. После должной отработки технологии и удешевления производства эти элементы сначала появлялись в ЭВМ среднего класса, а затем — в рабочих станциях, персональных компьютерах, современных микропроцессорах.

Рассмотрим краткую историю развития идей параллельной обработки в архитектуре как ЭВМ ведущих мировых производителей, так и отечественной вычислительной техники.

Первым массово применяемыми элементами параллельной обработки стали внедренные в ЭВМ **IBM 701** (1953) и **IBM 704** (1955) **разрядно-параллельная память и разрядно-параллельная арифметика**. Пионеры вычислительной техники (EDSAC, EDVAC, UNIVAC) имели разрядно-последовательную память, из которой слова считывались последовательно бит за битом. Указанные машины фирмы IBM стали первыми коммерчески доступными компьютерами, в которых были применены разрядно-параллельные память и арифметика. Очень популярная IBM 704 (изготовлено и продано свыше 150 экземпляров) имела при этом память на ферритовых сердечниках и аппаратное АЛУ с плавающей точкой.

Следующий виток развития параллельной обработки пришелся на 1958 г., когда к ЭВМ IBM 704 присоединили 6 **независимых процессоров ввода-вывода**. Поскольку скорость работы самого быстрого внешнего устройства (магнитной ленты) была на три порядка меньше скорости работы процессора, то последний неизбежно простаивал во время выполнения операций ввода-вывода. Независимые процессоры ввода-вывода получали специальные команды от основного процессора, после чего могли работать параллельно с ним. Новая ЭВМ, получив название IBM 709, оказалась очень удачной. Всего было продано около 400 экземпляров, причем последний был выключен только в 1975 г.!

Следующим шагом явилось создание в 1961 г. все той же фирмой IBM совместно с Лос-Аламосской научной лабораторией ЭВМ IBM STRETCH. В данном компьютере впервые были применены **опережающий просмотр вперед** для выборки команд и **расслоение памяти** на два банка для согласования низкой скорости выборки из памяти и скорости выполнения вычислений основным процессором.

В 1963 г. коллектив Манчестерского университета разработал ЭВМ ATLAS, в которой впервые был использован **конвейерный принцип выполнения команд**. Выполнение команды разбивалось на 4 ступени: выборка команды, вычисление адреса операнда, выборка операнда и выполнение команды. Конвейеризация позволила уменьшить время выполнения команды почти в 4 раза. Заметим, что кроме конвейера команд для данной ЭВМ впервые была разработана мультипрограммная ОС, основанная на использовании виртуальной памяти и системы прерываний.

Значительным событием стало появление в 1964 г. ЭВМ **CDC 6600** — разработки американской фирмы Control Data Corporation (CDC). В данной ЭВМ впервые использовалось **несколько независимых функциональных устройств**. Па-

раметры данного компьютера были настолько впечатляющими, что имеет смысл привести их и сегодня:

- время такта 100 нс;
- производительность 2–3 млн операций в секунду;
- оперативная память — 32 банка по 4096 60-ти разрядных слов;
- цикл памяти 1мкс;
- 10 независимых функциональных устройств;
- для ввода данных использовалась 7-ми канальная магнитная лента.

Система команд CDC 6600 предвосхитила появление RISC-архитектуры, а для охлаждения системы использовался фреон. Главным конструктором машин CDC являлся выдающийся американский ученый и инженер **Сеймур Роджер Крей** (Seymour R. Cray) (1925–1996), научную и инженерную деятельность которого трудно переоценить. Фактически именно его идеи и разработки определили архитектуру супер-ЭВМ в 60–80-х годах прошлого века, более того, они являются актуальными и по сей день.

Машины CDC 6600 имели огромный успех и практически вытеснили мэйнфреймы IBM с рынка высокопроизводительных ЭВМ для научных расчетов. Успех был закреплен выпуском в 1969 г. следующей модели — **CDC 7600** с восемью **независимыми конвейерными функциональными устройствами**, сочетавшими принципы параллельной и конвейерной обработки. Производительность CDC 7600 достигала 15 MIPS.

Следующей вехой на пути развития параллельных вычислений следует считать появление в 1974 г. **матричных процессоров (ILLIAC IV** фирмы Burroughs). Проект предполагал объединение 256 процессорных элементов (4 матрицы-квадранта по 64 ПЭ) в единую систему с производительностью до 1 GFLOPS. Несмотря на частичную реализацию (лишь один квадрант и 200 MFLOPS производительности), проект ILLIAC IV оказал огромное влияние на архитектуру ЭВМ и ВС последующих поколений.

В 1972 г. С. Крей покидает CDC и основывает свою знаменитую компанию Cray Research, которая в 1976 г. выпускает первый **векторно-конвейерный** компьютер **Cray-1** производительностью 160 MFLOPS. ЭВМ содержала 12 конвейерных функциональных устройств, способных обрабатывать **векторные** команды, работающие с массивами независимых данных.

Как и предыдущие машины С. Крея, Cray-1 имел значительный успех, который был закреплен появлением в 1985 г. ЭВМ **Cray-2** — первого компьютера, преодолевшего гигафлопный рубеж производительности. У Cray-2 данный показатель составил 1,2 GFLOPS. В этом суперкомпьютере в полной мере было использовано совмещение векторно-конвейерной и **мультипроцессорной** обработки — система состояла из набора взаимодействующих векторных процессоров.

На рубеже 80–90-х годов XX в. произошло то, что некоторые исследователи называют **первой суперкомпьютерной революцией**. В 1989 г. фирма Intel выпу-

стила серийный 64-разрядный микропроцессор i860, равный по производительности Cray-1. Начался массовый выпуск микропроцессоров, и с этих пор построение высокопроизводительных ВС стало продвигаться по пути комплексирования имеющихся на рынке готовых продуктов.

В этой связи особо следует отметить появление на рынке в 1985 г. **транспьютеров** британской фирмы Inmos. Данный микропроцессор имел RISC-архитектуру, позволявшую исполнять большинство команд за один такт. Кроме этого, каждый транспьютер имел по 4 линка — специальных высокоскоростных каналов связи. С помощью линков транспьютеры могли объединяться друг с другом, что положило начало созданию **массово-параллельных вычислительных систем (Massively Parallel Processing — MPP)**.

Массово-параллельные ВС представляют собой совокупность вычислительных модулей, объединенных высокоскоростными каналами связи. Каждый модуль содержит минимум один вычислительный процессор, собственную оперативную память, коммуникационное оборудование и управляется отдельной ОС. В течение 10 лет MPP-системам не было равных в производительности, что отражалось их лидирующими позициями в ведущемся с 1993 г. списке 500 самых мощных компьютеров мира (так называемом Top500). Среди высших достижений следует отметить построение в 1997 г. фирмой Intel первой терафлопной ЭВМ **ASCI Red**, объединившей до 9 тыс. процессоров, а также серийные массово-параллельные системы **SP2** компании IBM.

Альтернативой MPP с начала 90-х годов стала архитектура многопроцессорных ВС с общей памятью — **симметричный мультипроцессор (Symmetrical MultiProcessor — SMP)**. Несмотря на существенное ограничение на число объединяемых процессоров (не более 128), SMP-системы показывают наибольшую удельную производительность и являются гораздо проще в управлении (с помощью единой ОС) и программировании.

Стоит отметить, что перед разработчиками параллельных ВС в конце 80-х — середине 90-х годов XX в. стояли две основные технологические проблемы — построение вычислительных модулей и организация сбалансированных информационных коммуникаций между ними. Стоимость таких проектов измерялась десятками миллионов долларов. Однако резкий рост производительности персональных компьютеров, а также повсеместное внедрение сетевых технологий (100 Мбит Ethernet), привели к концу XX в. ко **второй суперкомпьютерной революции** — появлению **кластерной архитектуры**.

Кластерная ВС, как и MPP, представляет собой совокупность вычислительных модулей, объединенных коммуникационной средой. Разница состоит в том, что компонентами кластера являются не специализированные разработки, а *массово выпускаемые стандартные промышленные комплектующие*, в том числе персональные компьютеры и сетевое оборудование Ethernet. Основным достоинством подобных систем является наивысшее соотношение «производительность/стоимость».

Первым кластером стал проект **Beowulf**, реализованный в 1994 г. в NASA, а в 1998 г. и произошло собственно революционное событие — кластер **Avalon** (48 GFLOPS), объединивший 68 процессоров Alpha, стоимостью всего 152 тыс. долларов вошел в список Top500, заняв 114-е место и опередив многочисленные MPP- и SMP-системы с многомиллионной стоимостью.

С этого момента кластерный принцип построения постепенно стал доминировать в архитектуре параллельных ВС, в том или ином виде проявляясь в разработках супер-ЭВМ ведущих мировых производителей.

Практически все перечисленные этапы развития параллелизма, хотя и с небольшим опозданием, прошла отечественная вычислительная техника.

В 1957 г. в Институте точной механики и вычислительной техники АН СССР (ИТМ и ВТ) под руководством выдающегося советского ученого академика С. А. Лебедева завершились разработки ЭВМ общего назначения **БЭСМ-2**. Данная ЭВМ, как и IBM 701, имела параллельную систему обмена кодами между арифметическим и оперативным запоминающими устройствами. Машин в период до 1962 г. было выпущено несколько десятков экземпляров, и применялись они в вычислительных центрах и научно-исследовательских организациях для численного решения широкого круга математических задач. Преемницей БЭСМ-2 стала полностью полупроводниковая ЭВМ БЭСМ-4 (1961).

В 1964 г. в Конструкторском бюро Госкомитета по радиоэлектронике (впоследствии — КБ промышленной автоматики, а с 1978 г. — НИИ «Квант») под руководством В. С. Полина и В. К. Левина была разработана ЭВМ **«Весна»** производительностью до 300 тыс. операций в секунду. Данная машина имела центральный и периферийный процессоры, которые имели системы прерываний программ, необходимые для управления параллельной работой процессоров и внешних устройств (в мультипрограммном режиме), а также систему защиты памяти (управляемое разграничение доступа к памяти). В «Весне» применялось совмещение (параллелизм) обработки на всех структурных уровнях системы — АЛУ, память, команды в процессорах, внешние устройства.

В 1966 г. советские ученые Э. В. Евреинов и Ю. Г. Косарев опубликовали работу «Однородные универсальные вычислительные системы высокой производительности», в которой фактически излагались основы построения современных массово-параллельных ВС. Эта работа минимум на 20 лет опередила время.

В 1967 г. вступила в строй гордость отечественной вычислительной техники — знаменитая универсальная ЭВМ **БЭСМ-6**, созданная также в ИТМ и ВТ под руководством академика С. А. Лебедева. Всего было выпущено более 350 экземпляров этой поистине эпохальной ЭВМ, разработки и выпуск которой закончились только в 1987 г.!

В структуре БЭСМ-6 впервые в отечественной практике и независимо от зарубежных ЭВМ (STRETCH фирмы IBM) был широко использован принцип совмещения выполнения команд (до 14 одноадресных машинных команд могли на-

ходиться на разных стадиях выполнения). Этот принцип, названный академиком С. А. Лебедевым принципом «водопровода», стал впоследствии широко использоваться для повышения производительности универсальных ЭВМ, получив в современной терминологии название конвейера команд.

Работа модулей оперативной памяти, устройства управления и арифметико-логического устройства осуществлялась параллельно и асинхронно благодаря наличию буферных устройств промежуточного хранения команд и данных. Для ускорения конвейерного выполнения команд в устройстве управления были предусмотрены отдельная регистровая память хранения индексов, отдельный модуль адресной арифметики, обеспечивающий быструю модификацию адресов с помощью индекс-регистров, включая режим стекового обращения.

Ассоциативная память на быстрых регистрах (фактически кэш-память) позволяла автоматически сохранять в ней наиболее часто используемые операнды и тем самым сократить число обращений к оперативной памяти. Расслоение оперативной памяти обеспечивало возможность одновременного обращения к разным ее модулям из разных устройств машины.

Механизмы прерывания, защиты памяти, преобразования виртуальных адресов в физические и привилегированный режим работы для ОС позволили использовать БЭСМ-6 в мультипрограммном режиме и режиме разделения времени.

Следующим значительным достижением отечественной вычислительной техники стали многопроцессорные вычислительные комплексы (МВК) «**Эльбрус**» (1979 г. — «Эльбрус-1», 1984 г. — Эльбрус-2»). Комплексы были разработаны в ИТМ и ВТ под руководством академика В. С. Бурцева. Производительность «Эльбрус-1» — до 15 млн операций в секунду, «Эльбрус-2» — до 125 млн операций в секунду, емкость памяти «Эльбрус-1» — до 1 Мслов, «Эльбрус-2» — до 16 Мслов.

В состав комплексов входили до 10 центральных процессоров (на общей памяти), а также была предусмотрена возможность использования наряду с универсальными специализированных процессоров: СВС — для реализации прикладных программ, написанных для ЭВМ БЭСМ-6, и БПФ — для быстрого преобразования Фурье; процессоров ввода-вывода (до 4); процессоров приема-передачи данных (до 16) и др.

Развитием идей многопроцессорной обработки стала система «**Эльбрус 3-1**» (1989 г., ИТМ и ВТ, руководитель проекта — А. А. Соколов).

«Эльбрус 3-1» структурно представлял собой модульную систему, включавшую в себя несколько подсистем. Основные элементы процессорной подсистемы — модульный конвейерный процессор (МКП) производительностью до 550 MFLOPS, массовая память, подсистема высокоскоростной дисковой памяти и подсистема внешних устройств.

Связующим ядром, объединявшим различные компоненты в единое целое, являлась система коммутации и передачи данных, к которой в зависимости от

конкретной реализации могло быть подключено до 64 компонентов. Коммутационная система называлась также системным каналом, а объединяемые ею компоненты — его абонентами. В частности, МКП — один из абонентов.

ВС «Эльбрус 3-1» можно рассматривать как единую многопроцессорную систему с параллельной обработкой общих данных, распределенных по модулям локальной и массовой памяти. При этом для любой физической структуры системы была обеспечена связь абонентов по принципу «каждый с каждым».

Векторная обработка в МКП осуществлялась итеративными операциями над наборами упорядоченных данных — векторами, размещенными в локальной памяти. Эта обработка дополнялась более медленными операциями переупорядочения расположения векторов в локальной памяти.

После распада СССР, по времени совпавшего с первой суперкомпьютерной революцией, в России не прекратились разработки передовой вычислительной техники. Весомым достижением стало появление в 1994 г. массово-параллельной ВС **МВС-100** (разработка НИИ «Квант»). МВС-100 была построена на базе микропроцессоров фирмы Intel i860, объединяемых с помощью транспьютеров Т-805. В максимальной конфигурации (которой, правда, не удалось построить) система объединяла до 1024 процессоров и достигала производительности 100 GFLOPS (отсюда и цифра в названии).

Развитием идей МВС-100 явилась система **МВС-1000** (1998 г., НИИ «Квант»), также построенная по массово-параллельному принципу, но уже на базе вычислительного процессора DEC Alpha-21164. Производительность в максимальной конфигурации из 1024 процессоров — 1 TFLOPS. Опять-таки достичь максимальной конфигурации не удалось, но терафлопный рубеж был преодолен отечественными разработчиками в 2000 г. с выпуском кластерной ВС **МВС-1000М**, успешно проработавшей 4 года в Межведомственном суперкомпьютерном центре РАН (МСЦ РАН).

Наконец, силами НИИ «Квант» и МСЦ РАН в 2004 г. была введена в эксплуатацию кластерная система **МВС-15000** производительностью 10 TFLOPS.

5.4. ПОНЯТИЕ О ПАРАЛЛЕЛЬНОМ ПРОГРАММИРОВАНИИ

5.4.1. Парадигмы параллельного программирования

Как уже было отмечено, **параллельная программа** — это программа, содержащая части, называемые **ветвями**, которые могут выполняться одновременно на разных обрабатывающих устройствах (процессорах). Процесс написания подобных программ называют **параллельным программированием**.

При любом подходе к составлению параллельной программы программист вынужден, приступая к решению конкретной прикладной задачи, решить, хотя бы в самых общих чертах, **три вопроса**:

• как будут распределены данные по обрабатывающим устройствам (процессорам);

• как будет распределена по обрабатывающим устройствам (процессорам) обработка данных;

• в какие моменты данные из одних обрабатывающих устройств потребуются в других и, следовательно, когда их туда передавать.

Совокупность решений этих вопросов часто называют **схемой параллелизма** для данной задачи. Построение схемы параллелизма является целиком и полностью задачей программиста, а не инструментальной системы. Решив данную задачу, программист приступает к записи тем или иным образом построенной схемы параллелизма.

Существует много способов записи схемы параллелизма, различающихся не только формой записи (*программным интерфейсом*), но и самим набором понятий, которыми оперирует программист, разрабатывая параллельную программу. Набор данных понятий называют **парадигмой параллельного программирования**. В рамках парадигмы для разработки программ используют **модели программирования**. Отметим, что разделение на парадигмы и модели является довольно условным (в литературе часто можно встретить, что та или иная модель называется парадигмой, и наоборот), но мы считаем, что для обучения такое разделение целесообразно.

В настоящее время существуют две основных парадигмы распараллеливания вычислений: **параллелизм данных** и **параллелизм задач**. Рассматриваемые парадигмы программирования различаются методами записи схемы параллелизма.

Основная идея подхода, основанного на **параллелизме данных**, заключается в том, что одна операция выполняется сразу над всеми элементами массива данных. Различные фрагменты такого массива обрабатываются на векторном процессоре или на разных процессорах параллельной машины. Функции распределения данных между процессорами выполняются программно. Векторизация или распараллеливание, в этом случае чаще всего выполняется уже на этапе компиляции — перевода исходного текста программы в машинные команды. Схему параллелизма программист задает с помощью специальных директив компилятору, а также опций векторной или параллельной оптимизации. Для реализации данной парадигмы обычно используются специализированные языки программирования для описания параллельных вычислений.

К особенностям данного подхода относятся:

• централизованное программное управление обработкой данных;

• глобальное пространство имен в программе, т. е. для программиста существует одна единственная память, а детали структуры данных, доступа к памяти и межпроцессорного обмена данными от него скрыты.

В рамках данного подхода от программиста формально не требуется больших усилий по векторизации или распараллеливанию вычислений. Тем не менее прак-

тика показывает, что если программист плохо представляет себе, каким образом обрабатываемые данные будут размещены по процессорам, как будет организована их обработка и информационный обмен между процессорами, то эффективность разрабатываемых параллельных программ будет крайне низка.

Парадигма программирования, основанная на **параллелизме задач**, подразумевает, что вычислительная задача разбивается на несколько относительно самостоятельных подзадач (ветвей), и каждый процессор загружается своей собственной подзадачей. Компьютер при этом, как правило, представляет собой MIMD-машину.

Для каждой подзадачи пишется своя собственная программа на последовательном языке программирования, таком, как Fortran, C или на языках параллельного программирования, например, OCCAM. Чем больше подзадач, тем большее число процессоров можно использовать, тем большего ускорения вычислений можно добиться.

Существенным является то, что все эти программы должны обмениваться результатами своей работы. На практике такой обмен осуществляется вызовом процедур специализированной библиотеки. Программист при этом может контролировать распределение данных между процессорами и подзадачами и обмен данными.

По сравнению с подходом, основанном на параллелизме данных, такой подход более трудоемкий, так как в этом случае требуется определенная работа для того, чтобы обеспечить эффективное совместное выполнение различных программ.

Данному подходу свойственны следующие недостатки:

- повышенная трудоемкость разработки и отладки программы;
- на программиста ложится вся ответственность за равномерную загрузку процессоров параллельного компьютера, за минимизацию обменных операций между задачами;
- возможно возникновение тупиковых ситуаций (дедлоков и ливлоков).

Преимуществом данного подхода являются большие гибкость и свобода, предоставляемые программисту в разработке программы. Это позволяет эффективно использовать ресурсы параллельного компьютера и, как следствие, достичь максимального быстродействия. Практика показывает, что выигрыш в быстродействии может достигать нескольких раз по сравнению с подходом, основанном на параллелизме данных.

5.4.2. Модели параллельного программирования

В рамках парадигм программирования различают **модели параллельного программирования**. Рассмотрим сначала модели, основанные на параллелизме задач.

Модель «Процесс/канал» («Process/Channel»). Данная модель характеризуется следующими свойствами:

1) параллельное вычисление осуществляется одним или более одновременно исполняющихся процессов, число которых может изменяться в течение времени выполнения программы;

2) процесс — это последовательная программа с локальными данными. Процесс имеет входные и выходные порты, которые служат интерфейсом к среде процесса;

3) в дополнение к обычным операциям процесс может выполнять следующие действия: послать сообщение через выходной порт, получить сообщение из входного порта, создать новый процесс и завершить процесс;

4) операция передачи сообщения асинхронная — она завершается сразу, не ожидая того, когда данные будут получены. Операция приема сообщения синхронная: она блокирует процесс до момента поступления сообщения;

5) пары из входного и выходного портов соединяются очередями сообщений, называемыми **каналами** (channels). Каналы можно создавать и удалять. Ссылки на каналы (порты) можно включать в сообщения, так что связность графа межпроцессорных связей может изменяться динамически;

6) процессы можно распределять по физическим процессорам произвольными способами, причем используемое отображение (распределение) не оказывает влияния на семантику программы. В частности, множество процессов можно отобразить и на один процессор.

Понятие процесса позволяет говорить о местоположении данных: данные, содержащиеся в локальной памяти процесса, расположены «близко», прочие данные — «удаленно». Понятие канала обеспечивает механизм указания на то, что для продолжения вычислений одному процессу требуются данные другого процесса, т. е. между процессами имеется зависимость по данным.

Модель «Обмен сообщениями» («Message Passing»). В соответствии с этой моделью программы, написанные на традиционном последовательном языке программирования, исполняются процессорами компьютера. Программы обмениваются между собой данными, используя подпрограммы приема/передачи данных некоторой коммуникационной системы.

На сегодняшний день модель «Обмен сообщениями» является наиболее широко используемой моделью параллельного программирования. Программы этой модели, подобно программам модели «Процесс/канал», создают множество процессов, с каждым из которых ассоциированы локальные данные. Каждый процесс идентифицируется уникальным именем. Процессы взаимодействуют, посылая и получая сообщения. В этом отношении модель «Обмен сообщениями» является разновидностью модели «Процесс/канал» и отличается только механизмом, используемым при передаче данных. Например, вместо отправки сообщения в канал «channel 2» можно послать сообщение процессу «process 3».

Модель «Обмен сообщениями» не накладывает ограничений ни на динамическое создание процессов, ни на выполнение нескольких процессов одним процессором, ни на использование разных программ для разных процессов. Формальные описания систем обмена сообщениями не рассматривают вопросы, связанные с манипулированием процессами, однако при реализации таких систем обычно вводятся некоторые ограничения. Так, большинство систем обмена сообщениями при запуске параллельной программы создает фиксированное число идентичных процессов и не позволяет создавать и разрушать процессы в течение работы программы.

В таких системах каждый процесс выполняет одну и ту же программу (параметризованную относительно идентификатора либо процесса, либо процессора), но работает с разными данными. Реализуемая в этом случае модель вычислений получила название **SPMD** (Single Program — Multiple Data, «одна программа — много данных»). SPMD-модель приемлема и достаточно удобна для широкого диапазона приложений параллельного программирования, но она затрудняет разработку некоторых типов параллельных алгоритмов.

Используемая модель «Обмен сообщениями» бывает двух типов. Различают *модель Хоара* и *теговую модель*.

Механизм взаимодействия параллельных процессов, основанный на **модели Хоара**, имеет следующие особенности. Длины передаваемого и принимаемого сообщения должны быть согласованы на передающей и принимающей стороне. Информационный обмен реально начинается только после того, как отправитель явно выставил запрос **send** («послать сообщение»), а приемник — запрос **receive** («принять сообщение»). Сообщения доходят до принимающей стороны строго в порядке, в каком были отправлены передающей стороной.

Данный механизм позволяет организовывать высокоэффективные информационные обмены, однако его применение связано с существенными трудностями при программировании.

Более гибкой является **теговая модель** взаимодействия параллельных процессов. При посылке отправитель снабжает каждое сообщение специальным атрибутом — **тегом**, определяющим тип посылаемого сообщения. Получатель имеет возможность в каждый момент времени принимать сообщения нужного ему типа. Естественно, что при этом сообщения могут доходить до адресата не в том порядке, в котором были посланы. Данная модель более удобна для программирования, но имеет, как правило, худшие характеристики по быстродействию по сравнению с моделью Хоара за счет более громоздкой реализации.

Модель «Общая память» («Shared Memory»). В модели программирования с общей памятью, все процессы совместно используют общее адресное пространство, к которому они асинхронно обращаются с запросами на чтение и запись. В таких моделях для управления доступом к общей памяти используются механизмы синхронизации — семафоров блокировки процессов. Преимущество этой мо-

дели с точки зрения программирования состоит в том, что понятие собственности данных (монопольного владения данными процессом) отсутствует, следовательно, нет необходимости явно задавать обмен данными между производителями и потребителями. Эта модель, с одной стороны, упрощает разработку программы, но с другой стороны, затрудняет обеспечение локальности данных. В основном эта модель используется при программировании мультипроцессорных вычислительных систем с общедоступной памятью.

В рамках подхода, основанного на параллелизме данных, используется единственная модель — **«Параллелизм данных»** («Data Parallel»). В этой модели единственная программа задает распределение данных между всеми процессорами компьютера и операции над ними. Название модели происходит от того, что она эксплуатирует параллелизм, который заключается в применении одной и той же операции к множеству элементов структур данных. Распределяемыми данными обычно являются массивы. Как правило, языки программирования, поддерживающие данную модель, допускают операции над массивами, позволяют использовать в выражениях целые массивы, вырезки из массивов. Распараллеливание операций над массивами, параллельное выполнение циклов обработки элементов массива позволяет существенно повысить производительность программы. Каждый процессор выполняет обработку того подмножества элементов массива, которое расположено в его локальной памяти. Программы с параллелизмом данных могут быть оттранслированы и исполнены как на MIMD-, так и на SIMD-компьютерах.

Поскольку операции над каждым элементом данных можно рассматривать как независимые процессы, то степень детализации таких вычислений очень велика, а понятие «локальности» (распределения по процессам) данных отсутствует. Поэтому, компиляторы языков с параллелизмом данных требуют, чтобы программист предоставил информацию относительно того, как данные должны быть распределены между процессорами, другими словами, задал схему параллелизма. Обычно компилятор транслирует программу с параллелизмом данных в SPMD-программу, генерируя коммуникационный код автоматически.

5.4.3. Понятие о векторизации программ

Рассмотрим второй уровень параллелизма — параллелизм на уровне циклов. Данный уровень соответствует **векторной обработке** данных, т. е. обработке данных, представленных в виде векторов. Под **вектором** обычно понимают некоторый *упорядоченный* набор данных, размещенных в памяти *регулярным образом*. Примерами векторов в программах могут служить строки, столбцы, диагонали, массивы целиком, в то же время, например, поддиагональная часть двумерной матрицы вектором не является.

Некоторый фрагмент программы может быть обработан в векторном режиме, если для его выполнения могут быть использованы векторные команды. Поиск

таких фрагментов в программе и их замена на векторные команды называется **векторизацией программы**. **Векторные команды** — такие команды, которые выполняются над всеми элементами вектора одновременно. В противоположность векторным командам команды, работающие с обычными данными (не векторами), называются **скалярными**.

Типичным векторизуемым фрагментом программы может служить цикл с независимыми итерациями, например:

```
for (i = 0; i < N; ++i)
  A[i] = B[i] + C[i];
```

Выполнение данного цикла на скалярном процессоре будет выглядеть следующим образом — на каждой итерации цикла будут производиться действия:

- выборка из памяти операнда `B[i]`;
- выборка из памяти операнда `C[i]`;
- сложение `B[i]` и `C[i]`;
- запись в память по адресу `A + i` результата сложения.

Число итераций в этом случае будет равно `N`.

Теперь допустим, что указанная программа будет выполнена на процессоре, поддерживающем команды, оперирующие с векторами-массивами данных длиной `L`. Итерации цикла из нашего примера примут такой вид:

- выборка из памяти векторного операнда, содержащего элементы с `B[i]` по `B[i+L]`;
- выборка из памяти векторного операнда, содержащего элементы с `C[i]` по `C[i+L]`;
- сложение векторных операндов;
- запись в память по адресам с `A + i` по `A + i+ L` результата сложения.

Всего понадобится не более чем $\frac{N}{L}+1$ итерация цикла.

Задача векторизации программ возлагается на компилятор, транслирующий программу для исполнения на векторном процессоре. Следует заметить, что далеко не каждый цикл может быть векторизован. Основными препятствиями для векторизации могут являться следующие условия:

1. Зависимость итераций по данным. Примером может служить следующий фрагмент:

```
for( i = 1; i < Imax; ++i)
  A[i] = B[i] + A[i-1];
```

В данном цикле в каждой итерации в качестве операнда используется значение, вычисленное в предыдущей итерации. Зависимости по данным могут быть и между разными операторами тела цикла.

2. Отсутствие регулярно расположенных векторов. В следующем примере индекс массива для второго операнда вычисляется на каждой итерации цикла:

```
for( i = 0; i < Imax; ++i)
  {
   j = calc_index(i);
   A[i] = B[j] + A[i];
  }
```

Кроме этого, векторизация невозможна:

• когда обращения к объектам выполняются с помощью указателей, а не индексов массива;

• когда индексация массива осуществляется косвенно через другой массив, что имеет место при работе с разреженными массивами.

3. Вызов неизвестных подпрограмм и функций:

```
for( i = 0; i < Imax; ++i)
  func(i);
```

Следует заметить, что в некоторых случаях зависимости по данным в процессе анализа кода могут быть найдены и устранены. При этом проверка зависимостей может оказаться очень сложной, но в принципе возможной во время компиляции. В общем же случае задача определения действительного наличия зависимостей является NP-полной. Однако на практике многие частые случаи могут быть точно проанализированы при вполне умеренных затратах.

Кроме определения наличия зависимостей, компилятор старается также классифицировать тип зависимости. Это позволяет компилятору распознать зависимости по именам и устранить их путем переименования и копирования. Рассмотрим основные типы зависимостей и способы их устранения.

1. Истинные зависимости по данным между итерациями цикла. Например, возьмем следующий цикл:

```
for( i = 1; i <= Imax; i++ )
  {
   A[i+1] = A[i] + C[i];    /* оператор S1 */
   B[i+1] = B[i] + A[i+1]; /* оператор S2 */
  }
```

Если предположить, что A, B и C являются неперекрывающимися массивами, то между операторами этого цикла можно обнаружить две истинных зависимости:

1) операторы `S1` и `S2` используют значение, которое было вычислено на более ранней итерации, поскольку итерации *i* вычисляют `A[i + 1]` и `B[i + 1]`, которые считываются в итерации `[i + 1]`;

2) оператор `S2` использует значение `A[i + 1]`, которое вычисляется оператором `S1` в той же самой итерации.

Выявленные две зависимости отличаются друг от друга и имеют различный эффект. Рассмотрим зависимость оператора `S1` от его более ранней итерации. Ее наличие означает, что между различными итерациями цикла существует зависи-

мость по данным. Кроме того, поскольку оператор S1 зависит от самого себя, то последовательные итерации оператора S1 должны выполняться упорядоченно. Нетрудно видеть, что подобная зависимость по данным между итерациями делает векторизацию цикла невозможной.

Вторая зависимость (оператора S2 от оператора S1) не передается от итерации к итерации. Поэтому при наличии подобного типа зависимостей можно сделать цикл векторизуемым при условии, что каждая пара операторов в итерации выполняется в определенном порядке. Покажем эту возможность на примере следующего цикла:

```
for( i = 1; i <= Imax; ++i )
  {
   A[i] = A[i] + B[i];    /* оператор S1 */
   B[i+1] = C[i] + D[i]; /* оператор S2 */
  }
```

В этом примере оператор S1 использует значение, которое присваивается оператором S2 в предыдущей итерации. Таким образом, здесь имеется зависимость между итерациями операторов S1 и S2.

Несмотря на имеющиеся зависимости, этот цикл можно сделать векторизуемым, так как здесь ни один из операторов S1 и S2 не зависит сам от себя. Поэтому указанный цикл можно представить в следующем виде:

```
A[1] = A[1] + B[1];
for( i = 1; i <= Imax-1; ++i)
  {
   B[i+1] = C[i] + D[i];
   A[i+1] = A[i+1] + B[i+1];
  }
B[Imax+1]=C[Imax]+D[Imax];
```

2. Истинные зависимости с расстоянием. Часто зависимости между итерациями цикла появляются в форме рекуррентного отношения, при этом некоторые рекуррентные отношения могут быть источником значительной степени параллелизма. Например, рассмотрим цикл:

```
for ( i = k+1; i <= Imax; ++i)
      Y[i] = Y[i-k] + Y[i];
```

На итерации j цикл обращается к элементу j - k. Говорят, что цикл имеет *зависимость с расстоянием* k. Чем больше расстояние, тем больше степень потенциального параллелизма, которую можно получить при помощи разворачивания цикла. Например, если мы разворачиваем цикл, имеющий зависимость с расстоянием 1, последовательные операторы зависят друг от друга. Если мы разворачиваем цикл, который имеет зависимость с расстоянием k, то имеется последовательность из k команд, которые не имеют зависимостей и могут быть векторизованы.

Хотя многие циклы с зависимостями между итерациями имеют расстояние зависимостей 1, случаи с большими расстояниями в действительности возникают

достаточно часто. Большее расстояние между зависимостями может обеспечивать большую степень векторизации.

Далее рассмотрим два типа так называемых *ложных (или псевдо-) зависимостей.*

3. Зависимости по выходу. В этом случае цикл содержит несколько последовательных операторов, изменяющих один и тот же элемент массива. Параллельное выполнение этих операторов может привести к конфликту вида «запись после записи».

Примером зависимости по выходу может являться следующий цикл:

```
for( i = 1; i <= Imax; ++i )
  {
   A[i] = B[i] + d;    /* оператор S1 */
   ...
   A[i] = C[i] / k;    /* оператор S2 */
  }
```

В данном примере имеется зависимость по выходу от S1 к S2. Для устранения зависимостей по выходу используется прием *переименования переменных.* Приведенный цикл изменяется следующим образом:

```
for( i = 1; i <= Imax; ++i )
  {
   D[i] = B[i] + d; /* A переименовывается в D */
   ...
   A[i]=C[i]/k; /* зависимость по выходу устранена*/
  }
```

4. Антизависимость проявляется, когда значение элемента массива, используемое в качестве операнда одним из операторов цикла, изменяется в последующих операторах, что делает невозможным параллельное исполнение этих операторов. Устранение антизависимости также осуществляется с помощью переименования переменных.

К качестве комплексного примера приведем следующий цикл, имеющий несколько типов зависимостей. Найдем все истинные зависимости, зависимости по выходу и антизависимости и устраним зависимости по выходу и антизависимости с помощью переименования.

```
for( i = 1; i <= Imax; ++i )
  {
   Y[i] = X[i] / c;    /* оператор S1 */
   X[i] = X[i] + c;   /* оператор S2 */
   Z[i] = Y[i] + c;   /* оператор S3 */
   Y[i] = c - Y[i];    /* оператор S4 */
  }
```

В этих четырех операторах имеются следующие зависимости:

1) истинные зависимости от S1 к S3 и от S1 к S4 из-за Y[i]. Эти зависимости приведут к ожиданию операторами S3 и S4 завершения оператора S1. Тем не менее отсутствуют истинные зависимости между различными итерациями цикла, что делает возможным его векторизацию;

2) антизависимость от S1 к S2 из-за X[i]. Значение X[i] изменяется в S2, но при этом используется в качестве операнда в S1;

3) зависимость по выходу от S1 к S4, так как Y[i] изменяется в обоих этих операторах.

Следующая версия цикла устраняет выявленные зависимости.

```
for (i = 1; i <= Imax; ++i )
  {
    /* Y переименовывается в T для устранения
    зависимости по выходу */
    T[i] = X[i] / c;
    /* X переименовывается в X1
    для устранения антизависимости  */
    X1[i] = X[i] + c;
    Z[i] = T[i] + c;
    Y[i] = c - T[i];
  }
```

После цикла массив X оказался переименованным в X1. В коде программы, следующем за циклом, компилятор может заменить имя X на имя X1. В данном случае переименование не требует операции копирования, а может быть выполнено с помощью заменяющего имени или соответствующего распределения регистров. Однако в других случаях переименование может потребовать копирования.

Анализ зависимостей является важнейшей технологией для повышения степени параллелизма и для обнаружения параллелизма уровня цикла является базовым инструментом. От этого анализа существенно зависит эффективность компиляции программ для векторных машин и мультипроцессоров.

5.4.4. Средства параллельного программирования

5.4.4.1. Языковые средства

Языковые средства параллельного программирования — это средства, позволяющие записать готовую схему параллелизма на том или ином языке программирования. Данные инструментальные программные средства разрабатываются как за рубежом, так и в России. Рассмотрим несколько примеров.

Язык **OCCAM** основан на модели «процесс/канал» и является *языком взаимодействующих процессов*. Каждый процесс, описанный в нем, выполняет последовательность действий и завершается. Действие может быть операцией присваивания, ввода или вывода. Присваивание изменяет значение некоторой переменной,

ввод получает значение из канала, вывод посылает значение в канал. Межпроцессные взаимодействия синхронизируются. Информация копируется из одного процесса в другой только после того, как оба процесса готовы к взаимодействию по одному и тому же каналу. Процесс может ждать данные из нескольких каналов и вводить их по мере поступления в каналы.

Язык OCCAM предназначен прежде всего для многопроцессорных систем, построенных на базе транспьютеров. К сожалению, широкого распространения данный язык не получил и на другие аппаратные платформы не переносился.

Одной из отечественных разработок является система программирования **GNS**, предназначенная для многопроцессорных вычислительных систем с распределенной памятью. Принципы построения системы GNS реализованы в виде расширений языков программирования Fortran (Fortran GNS) и C (C GNS).

Система программирования GNS вобрала в себя основные возможности, присущие различным библиотекам, реализующим модель Хоара. В языках GNS имеются развитые и удобные режимы передачи сообщений, допускающие совмещение счета с обменом информацией. При этом есть возможность в одном сообщении передавать несмежные участки памяти, что повышает эффективность информационных обменов.

Язык Fortran GNS является расширением языка Fortran-77 и базируется на концепции абстрактной GENESIS-машины. В соответствии с этой концепцией задача пользователя представляет собой множество подзадач, которые динамически создаются на указанных процессорах распределенной системы и взаимодействуют между собой путем передачи и приема сообщений в любом из трех режимов — синхронном, асинхронном и в режиме без ожидания. При этом взаимодействовать может любая пара подзадач.

Средства языка Fortran GNS были приняты за основу расширений языка C при разработке языка C GNS.

Отдельную группу языковых средств составляют языки, **трансляторы с которых автоматически преобразуют последовательную программу в параллельную**. Как правило, трансляция происходит из исходных текстов последовательной программы на языке высокого уровня в исходные тексты параллельного варианта этой же программы на этом же языке высокого уровня.

Наибольшее распространение получили параллельные расширения языка Fortran. В 1990 г. вышла в свет спецификация языка Fortran-90. Fortarn-90 (позже — Fortran-95) представляет собой развитие языка Fortran-77, включающее в себя элементы параллелизма, такие, как операции над массивами и секциями массивов и глобальные операции, а в Fortran-95 также и средства задания параллелизма циклов. Существуют множество реализаций данного языка для самых различных платформ.

Дальнейшим развитием Fortran-90 явился язык **HPF** (High Performance Fortran). В язык HPF включены богатые средства для распределения данных по про-

цессорам. Необходимые коммуникации и синхронизации реализуются компилятором. Часть расширений реализована в виде функций и операторов языка, а часть — в виде директив компилятору, которые являются комментариями языка Fortran. Практически все производители супер-ЭВМ включают в свою поставку компиляторы HPF.

Среди отечественных инструментальных средств автоматического распараллеливания программ следует отметить систему программирования **DVM**.

DVM-система предназначена для создания переносимых и эффективных вычислительных приложений на языках C-DVM, Fortran-DVM и Java-DVM для компьютеров с различной архитектурой.

Языки программирования (Fortran DVM, C-DVM) обеспечивают:

- глобальное пространство имен (единое адресное пространство);
- программирование в терминах поведения системы в целом;
- программирование без низкоуровневой детализации взаимодействия и синхронизации задач;
- использование одного варианта программы для последовательного и параллельного выполнения.

Высокая эффективность DVM-программ достигается за счет средств точного специфицирования параллельного выполнения программ, совмещения вычислений и обмена сообщениями в системах с распределенной памятью, а также средств для балансировки загрузки.

В рамках модели «общая память» разработан стандарт **Open MP**, позволяющий на базе этой модели создавать переносимое, масштабируемое программное обеспечение. Модель общей памяти Open MP основана на операторах разветвления (fork) и объединения (join) процессов. Выполнение программы начинается одним головным процессом. Оператор разветвления порождает группу параллельных процессов. После объединения из всех параллельных процессов остается только один головной процесс. Параллельные процессы могут, в свою очередь, выполнять операции разветвления, что приводит к вложенности групп параллельных процессов.

Open MP — гибкий стандарт, легко развиваемый применительно к любому языку последовательного программирования. Компонентами Open MP являются управляющие структуры, окружения данных, синхропримитивы и библиотеки времени исполнения. Так же, как и в языках HPF и DVM, OpenMP-инструкции вставляются как комментарии к языку программирования.

Рассматривая средства автоматического распараллеливания, необходимо упомянуть о **непроцедурных языках программирования**, компиляторы с которых формируют параллельный код. Примером подобного средства может служить непроцедурный язык **Норма**.

Непроцедурный язык Норма предназначен для записи численных методов решения задач математической физики разностными методами. Главная идея, по-

ложенная в основу языка Норма, заключается в том, что полученное прикладным специалистом описание решения задачи (расчетные формулы) почти непосредственно используется для ввода его в вычислительную систему и проведения расчетов.

Язык Норма дает прикладному математику возможность сформулировать свою задачу в привычных для него терминах. Организацию процесса вычислений с учетом архитектуры ЭВМ выполняет транслятор с языка Норма.

5.4.4.2. Коммуникационные библиотеки

При использовании парадигмы «параллелизм задач» очень часто программа пишется на одном из традиционных языков программирования. В зависимости от используемой модели, информационный обмен между ветвями параллельной программы осуществляется путем вызова той или иной подпрограммы. Набор объединенных подпрограмм, реализующих информационный обмен, будет представлять собой **коммуникационную библиотеку**. При этом разные библиотеки могут быть ориентированы на разные модели параллельного программирования.

Большинство производителей параллельных ВС или коммуникационных сред для них снабжают разрабатываемую ими систему своей **низкоуровневой библиотекой информационного обмена**. Примерами могут служить библиотеки **NX** (Intel Paragon и ASCI Red), **GM** (Myrinet), **OC Router** (МВС-100х). Поскольку основная задача низкоуровневых библиотек — достижение максимального быстродействия, они, как правило, реализуют модель Хоара. Существенным недостатком низкоуровневых библиотек, помимо трудностей программирования, является отсутствие единого стандарта на синтаксис и семантику используемых функций и подпрограмм.

Для обеспечения совместимости между параллельными ЭВМ различной архитектуры было разработано несколько стандартов, следование которым обеспечивает мобильность разрабатываемых параллельных программ. К числу наиболее распространенных следует отнести стандарты **PVM**, **MPI**. В настоящее время существуют множество различных реализаций указанных стандартов, многие из которых поставляются производителями параллельных ЭВМ.

Стандарты PVM и MPI соответствуют теговой модели «обмен сообщениями». Особенностью стандартов PVM и MPI является их функциональная мощность по сравнению с библиотеками примитивов информационного обмена. Фактически эти стандарты представляют собой универсальные инструменты для создания сложных параллельных приложений.

PVM (Parallel Virtual Machine — «параллельная виртуальная машина») представляет собой программное средство для объединения набора неоднородных вычислительных средств в единое целое. Для использующего PVM программиста такой набор средств представляется как единая виртуальная параллельная машина. Программист ничего не должен знать ни об архитектуре физических ЭВМ (количество процессоров, быстродействие, разрядность данных, состав перифе-

рийных устройств), ни об их соединении (тип и пропускная способность каналов связи, топология сети и т. д.). Общим для объединяемых с помощью PVM вычислительных систем является операционная система Unix с поддержкой протоколов TCP/IP.

В рамках PVM стандартизуется не только интерфейс программиста (набор и смысл доступных для вызова из программы функций), но и интерфейс пользователя, включающий команды управления параллельной виртуальной машиной. Командами управления можно менять состав виртуальной машины, подключать и отключать узлы. Можно программно запускать из одних выполняющихся в рамках виртуальной машины процессов другие, автоматически включающиеся в виртуальную машину, и использовать их в качестве ветвей параллельной программы. Процесс может отключиться от виртуальной машины и продолжить работу как обычный процесс пользователя в рамках ОС узла.

Программный интерфейс обмена сообщениями PVM довольно прост. Все посылки сообщений, если явно не задано противное, буферизующие: операция посылки обязана завершиться вне зависимости от того, выполнен ли соответствующий ей прием. Прием бывает ждущий (завершающийся только тогда, когда есть, что принимать) и не ждущий (способный сообщить, что принимать нечего, и тем самым завершающийся безусловно). Логика системы передачи сообщений PVM практически исключает появление тупиковых ситуаций (дедлоков), но не допускает эффективной реализации.

При посылке сообщения указывается его тег, при приеме сообщения можно «фильтровать» по тегу. Помимо обменов «точка-точка», имеется аппарат широковещания и редукции (посылка сообщения всем процессам или группе процессов, поиск глобального максимума по массивам, распределенным по разным процессам и т. п.).

Для обеспечения работы в гетерогенной среде, в состав которой входят ЭВМ с разным машинным представлением данных, в акт посылки и приема сообщений явно входит процесс упаковки и распаковки данных. Данные передаются в машинно-независимом формате.

PVM предназначена для решения задач с так называемым крупным зерном параллелизма (когда доля вычислительных операций многократно превышает долю коммуникационных) с не очень высокими требованиями к эффективности коммуникаций; для задач, в которых гибкость системы управления процессами и поддержка гетерогенности важнее быстродействия подсистемы передачи сообщений.

Другим стандартным средством, позволяющим организовывать параллельные вычисления, является **MPI** (Message Passing Interface — «интерфейс передачи сообщений»). MPI в отличие от PVM стандартизует лишь интерфейс программиста, т. е. набор функций и аргументов коммуникационной библиотеки. Одним из существенных достоинств MPI является возможность сочетания простоты и

эффективности при реализации стандарта и разработке параллельных программ на его основе. В частности, стандарт содержит разветвленный инструментарий функций пересылки данных, возможность согласования типов данных между ЭВМ различной архитектуры, а также конструирования собственных пользовательских типов данных. Последнее позволяет определить пользовательский тип таким образом, что использование его, например при передаче матрицы, приведет к транспонированию матрицы.

MPI основан на модели SPMD. Процессы, способные играть роль ветвей параллельной программы, порождаются в момент запуска программы все сразу. Если процесс MPI, пользуясь возможностями ОС, породит дочерний процесс, этот порожденный процесс новым процессом в рамках MPI быть не сможет. Порожденные процессы статически нумеруются, после чего возможны информационные обмены с помощью вызова подпрограмм коммуникационной библиотеки.

Гибкость, эффективность и большой объем возможностей, предоставляемые стандартом MPI, а также его широкое распространение делают реализации данного стандарта хорошей базой для разработчиков инструментальных средств более высокого языкового уровня.

5.4.4.3. Средства отладки, мониторинга и оптимизации параллельных программ

Отладка параллельных программ требует гораздо больших усилий от программиста, чем отладка последовательных программ. Это объясняется асинхронностью выполнения ветвей параллельных программ, а также более сложным восприятием человеком параллельных алгоритмов.

Отладка параллельных программ производится с помощью трассировки, ее автоматизированного анализа, а также автоматизированного анализа зафиксированных состояний выполнения программ. При этом важное значение имеют средства визуализации процесса отладки, позволяющие наглядно отобразить структуру параллельной программы, характеристики выполнения ее ветвей и их взаимодействие путем передачи сообщений.

Средства отладки и мониторинга могут быть использованы в следующих трех режимах:

- режиме выполнения программы на параллельной ЭВМ;
- режиме псевдовыполнения программы на специальной инструментальной ЭВМ. Псевдовыполнение программы осуществляется путем обработки трассировки, накопленной во время выполнения программы на параллельной ЭВМ;
- режиме имитации на инструментальной ЭВМ выполнения программы на параллельной ЭВМ.

Средства мониторинга (наблюдения за поведением параллельной программы) могут рассматриваться как составная часть средств отладки, поскольку они позволяют программисту глубже понять поведение своей программы. К подоб-

ным средствам относят средства сбора статистики и средства профилирования программ.

Большинство современных компиляторов для параллельных ЭВМ, как правило, включают в себя средства отладки и мониторинга. Одним из наиболее полных (в смысле функциональности) и универсальных средств отладки и мониторинга является система **Digital PSE (Parallel Software Environment)**.

Система Digital PSE была спроектирована фирмой DEC для поддержки разработки и выполнения параллельных приложений на SMP- и MPP-системах, а также кластеров из рабочих станций и серверов, работающих под управлением ОС Digital Unix. Система PSE представляет собой обширный инструментарий разработчика, включающий поддержку HPF, PVM и MPI. Непосредственная реализация PSE состоит из набора динамических библиотек, утилит и системных процессов.

Система PSE снабжена так называемым n-оконным отладчиком. Каждая ветвь параллельного приложения может контролироваться отдельной копией отладчика, которая имеет свой собственный терминал (окно). Пользователю доступны основные отладочные функции, такие, как установка точек прерывания, обратная трассировка, просмотр переменных. Кроме этого, отладчик PSE содержит ряд команд поддержки языка HPF, которые позволяют просматривать элементы распределенных массивов данных. К сожалению, область применения PSE ограничена вычислительными системами фирмы DEC (Compaq, HP).

Одним из наиболее мощных, на сегодняшний день, параллельных отладчиков является отладчик **TotalView** фирмы Etnus. Помимо традиционных возможностей *n*-оконной отладки и мощных функций визуализации TotalView предоставляет средства отладки мультипроцессных программ на одной машине, на многопроцессорной ЭВМ, на кластере однородных компьютеров. Пользователь имеет возможность отслеживать поведение любого процесса в распределенной вычислительной системе, просматривая данные с разной степенью детализации.

TotalView поддерживает языки C, C++, Fortran-77, Fortran-90, а также ассемблер, причем по желанию пользователя часть кода при отладке может быть представлена на языке высокого уровня, а часть — на ассемблере. TotalView имеет возможность осуществлять отладку в терминах MPI и PVM. В настоящее время фирма Etnus выпустила реализации TotalView для большинства распространенных аппаратных платформ.

К средствам мониторинга и оптимизации параллельных программ относятся **профилировщики**. Профилировщики представляют собой программные средства, позволяющие измерять время исполнения программ. В ходе работы программы, запущенной под управлением профилировщика, происходит накопление данных о ее выполнении, которые после их анализа представляются в виде отчета профилирования.

Профилирование отличается от большинства других шагов разработки программы, так как производится лишь после того, как программа отлажена и работа-

ет корректно. Обычно профилирование осуществляется в конце цикла разработки программного продукта, когда есть потребность детально изучить его работу и оптимизировать время исполнения.

Как и отладочные средства, профилировщики поставляются фирмами — производителями параллельных ЭВМ. Примером средств профилирования могут служить профилировщики уже упоминавшейся системы Digital PSE, а также средство **Convex CXpa** фирмы Hewlett-Packard, предназначенное для оптимизации программ в системе Convex Exemplar.

5.5. КЛАСС MISD — КОНВЕЙЕРНАЯ ОБРАБОТКА ИНФОРМАЦИИ

Конвейерная обработка информации основана на разделении подлежащей исполнению функции на более мелкие части, называемые **ступенями**, и выделении для каждой из них отдельного блока аппаратуры. Так, обработку любой машинной команды можно разделить на несколько этапов (несколько ступеней), организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

Как было уже отмечено, согласно классификации Флинна, ЭВМ и ВС, построенные по конвейерному принципу, относятся к классу MISD — систем с множественным потоком команд и одиночным потоком данных.

Конвейерная обработка может быть реализована в системе с одним процессором, разделенным на некоторое число последовательно соединенных операционных блоков, каждый из которых специализирован на выполнении строго определенного этапа арифметической или логической операции. Данный тип обработки данных иногда называется **арифметико-магистральной обработкой.**

Очевидно, что идея конвейера может быть распространена и на выполнение команд. Подобного типа обработка при помощи конвейера команд носит название **«командно-магистральная обработка».**

При применении конвейера команд цикл выполнения команды разбивается на ряд элементарных этапов, например: формирование адреса команды, выборка команды из памяти, расшифровка кода операции, формирование адреса операнда, выборка операнда из памяти и, наконец, выполнение самой арифметической или логической операции. В обрабатывающем устройстве предусматриваются блоки, которые независимо друг от друга и параллельно могут выполнять указанные этапы — из них и строится магистраль обработки.

Арифметико-магистральная и командно-магистральная обработки относятся к классу **микроконвейерной обработки.**

Разумеется, в вычислительных системах можно одновременно использовать и конвейер команд, и конвейер арифметических операций, и даже несколько парал-

лельно работающих конвейеров команд и арифметических операций. В последнем случае может быть получена очень высокая производительность системы.

Подробно особенности микроконвейерной обработки рассматриваются в курсе «Современные микропроцессоры», здесь же мы рассмотрим, как с помощью конвейерной обработки может быть организовано одновременное решение различных частей одной задачи.

Рассмотрим данный способ обработки применительно к задачам, программа решения которых должна многократно повторяться. Разобьем некоторую программу A на одинаковые по времени реализации части — подпрограммы $A_1, A_2, \ldots, A_n$, последовательное выполнение которых решает поставленную задачу. Теперь запрограммируем n процессоров $P_1, P_2, \ldots, P_n$ на решение этих подпрограмм и расположим их в цепочку-конвейер так, что результат работы i-го процессора является набором исходных данных для $(i+1)$-го процессора (рис. 5.10).

Пусть $X_1, X_2, \ldots, X_k$ — исходные данные k задач, которые должны быть решены программой A k-кратным ее повторением, а $R_1, R_2, \ldots, R_n$ — результаты расчетов подзадач $A_1, A_2, \ldots, A_n$.

Будем последовательно подавать на вход цепочки процессоров исходные данные $X_1, X_2, \ldots, X_k$ с интервалом времени $\frac{T}{n}$, где T — общее время решения одной задачи, если бы она выполнялась на одном процессоре (тогда $\frac{T}{n}$ — время решения одной подзадачи A_i на любом из процессоров цепочки). В этом случае, как легко заметить, в любой момент времени каждый процессор будет решать свою подзадачу: если первый — i-ю, то второй — $(i-1)$-ю, третий — $(i-2)$-ю, n-й процессор — $(i-n+1)$-ю. Легко подсчитать производительность такого конвейера. Время решения k задач на конвейере из n процессоров $T_{\text{кон.}}$

$$T_{\text{кон.}} = T + \frac{(k-1)\cdot T}{n} = \frac{(k+n-1)\cdot T}{n}. \qquad (5.8)$$

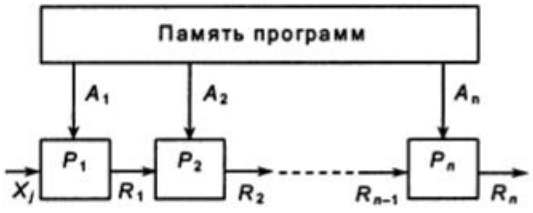


Рис. 5.10. Конвейер вычислений на j-м шаге

Один процессор затратил бы на это время $T_{\text{пос.}} = T \cdot k$. Таким образом, соотношение

$$\frac{T_{\text{пос.}}}{T_{\text{кон.}}} = \frac{n \cdot k}{k+n-1} \qquad (5.9)$$

определяет выигрыш, достигаемый в результате применения конвейера. Из этого выражения видно, что при фиксированном k ускорение обработки потока данных пропорционально числу n — длине конвейера, а при фиксированном n — длине обрабатываемой последовательности данных k.

Рассмотренный тип обработки называется **макроконвейерной обработкой** и используется на практике только для создания специализированных ВС, постоянно решающих одну и ту же задачу на большом потоке данных.

5.6. СИСТЕМЫ КЛАССА SIMD

5.6.1. Векторные ВС

Рассмотрим архитектуру векторной ВС на примере классических ЭВМ Cray, выпускавшихся с середины 70-х годов XX в. компанией Cray Research. В большинстве своем данные ЭВМ представляли собой компьютеры, объединяющие несколько векторно-конвейерных процессоров, работающих над общей памятью.

Оперативная память этих ВС разделяется всеми процессорами и секцией ввода-вывода. Для увеличения скорости выборки данных память разделена на множество банков, которые могут работать одновременно.

Все процессоры имеют одинаковую структуру (рис. 5.11), которую рассмотрим подробнее. В состав процессора входят три секции:

- функциональных устройств;
- регистров;
- управления программой.

Функциональные устройства, работающие в режиме конвейера, разбиты на 4 группы: адресная, скалярная, операции с плавающей запятой, векторная. Число конвейерных ступеней (на рис. 5.11 это число указано в скобках после названия устройства) в каждом из функциональных устройств зависит от сложности операций и сравнительно невелико (от 1 до 14), что обеспечивает быстрое время загрузки конвейеров.

Функциональные устройства Cray, играющие роль арифметико-логических преобразователей, не имеют непосредственной связи с главной памятью. Они имеют доступ только к быстрым операционным регистрам, из которых выбираются операнды и в которые помещаются результаты после выполнения соответствующих действий.

Три группы операционных регистров, непосредственно связанные с арифметико-логическими устройствами, называются основными. К ним относятся восемь так называемых ***A*-регистров**. *A*-регистры связаны с двумя функциональными модулями, выполняющими сложение (вычитание) и умножение целых чисел. Эти операции используются главным образом для преобразования адресов, их базирования и индексирования. Они также используются для организации счетчиков циклов. В ряде случаев *A*-регистры можно использовать для выполнения арифметических операций над целыми числами.

В следующую группу основных операционных регистров входят восемь 64-разрядных ***S*-регистров**, непосредственно связанных с функциональными устройствами выполнения арифметических действий со скалярными величинами, представленными с фиксированной и плавающей запятой.

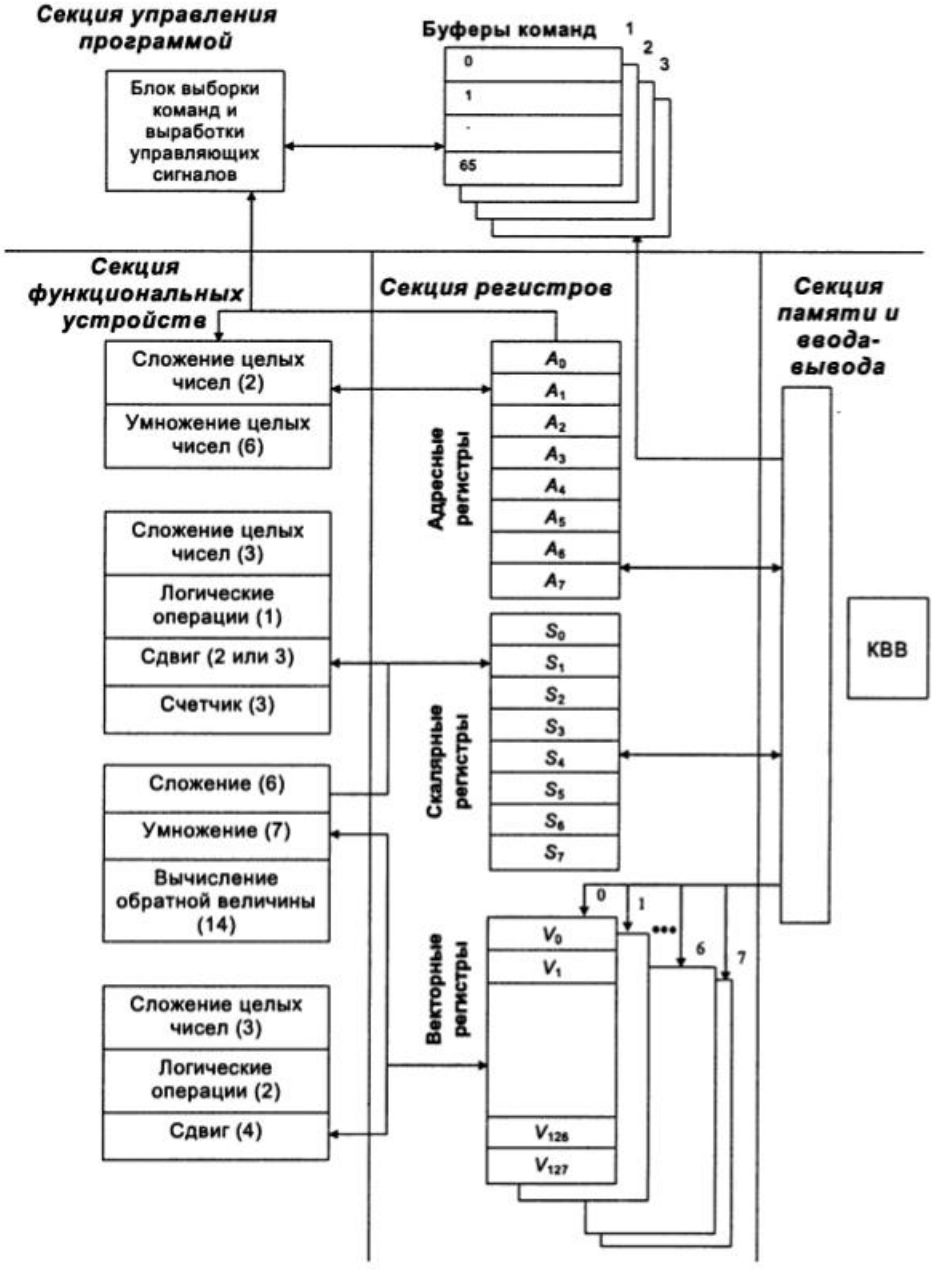


Рис. 5.11. Схема векторно-конвейерного процессора Cray

Особый интерес представляют восемь многоэлементных векторных ***V*-регистров**, которые предназначены для хранения восьми операндов-векторов. Каждый такой операнд состоит из нескольких (до 128) элементов. В свою очередь, каждый элемент представляет собой 64-разрядное слово — число с плавающей или фиксированной запятой.

В системе команд ЭВМ предусмотрены специальные операции, в качестве операндов которых выступают многоэлементные векторы. Поскольку не во всех задачах требуется обрабатывать векторы полной размерности, специальный управляющий регистр процессора задает требуемую размерность (число элементов). Этот регистр программно управляем, что позволяет в процессе вычислений изменять размерность обрабатываемых векторов. Кроме того, в центральном процессоре предусмотрен регистр маски, с помощью которого можно блокировать выполнение арифметико-логических действий над некоторыми элементами вектора, т. е. осуществлять выборочные поэлементные действия.

Все основные операции, выполняемые процессором (обращения в память, обработка команд и выполнение инструкций), конвейерные. Большинство функциональных устройств являются независимыми, поэтому несколько операций могут выполняться одновременно. Например, для операции $A = (B + C) \cdot D \cdot E$ порядок выполнения может быть следующим (считаем, что все аргументы загружены в *S*-регистры). Генерируются три инструкции: умножение *D* и *E*, сложение *B* и *C* и умножение результатов двух предыдущих операций. Первые две операции выполняются одновременно, затем третья.

Векторные операции, использующие различные ФУ и регистры, также могут выполняться параллельно. Архитектура Cray позволяет использовать регистр результатов векторной операции в качестве входного регистра для последующей векторной операции, т. е. выход сразу подается на вход. Это называется **зацеплением векторных операций**. Вообще говоря, глубина зацепления может быть любой, например чтение векторов, выполнение операции сложения, выполнение операции умножения, запись векторов.

Зацепление функциональных устройств и конвейеризацию вычислений можно продемонстрировать на следующем примере. В вычислительных методах линейной алгебры часто встречается процедура, состоящая в том, что строку матрицы (все элементы строки матрицы) умножают на некоторую скалярную величину и затем вычитают из элементов другой строки. Поскольку функциональные устройства, выполняющие операции умножения и вычитания, могут работать одновременно, то данная вычислительная процедура может быть исполнена следующим образом.

Первый элемент первого вектора умножается на скалярную величину, после чего начинается вычитание результата из первого элемента второго вектора. Пока происходит это вычитание, параллельно выполняется операция умножения скаляра на второй элемент первого вектора и т. д.

Система команд ЭВМ семейства Cray прямо отражает регистровую структуру центрального процессора, своеобразие связи функциональных устройств с операционными регистрами и связи их с оперативной памятью. Команды машин Cray двух форматов: короткие команды — 16 разрядов и длинные — 32 разряда. Семь первых разрядов определяют код операции, затем следуют трехразрядные поля, определяющие соответственно номер регистра результата и номера регистров исходных операндов.

В одном слове могут размещаться до четырех команд короткого формата. Длинные команды могут начинаться в одном слове и продолжаться в следующем. Это позволяет плотно упаковывать команды в памяти машины и в какой-то степени ускоряет их выборку.

Отметим, что защита памяти и система прерываний организованы так, что значительная доля работы по запоминанию информации для возврата, по организации процесса переключения с одной задачи на другую возлагается на операционную систему. Отсутствие достаточно развитых аппаратных средств динамического перераспределения памяти, упрощенный аппарат преобразования программных адресов в физические (аппаратное базирование) вполне допустимы в машинах, предназначенных для решения крупных научных задач. Этот режим «научных вычислений» характерен сравнительно редким появлением ситуаций, требующих переключения с решения одной задачи на другую. Это, в свою очередь, позволяет по-иному реализовать защиту и прерывания, не заботясь о быстром аппаратном их выполнении.

Для межпроцессорного взаимодействия служит специальная секция, которая содержит разделяемые регистры и семафоры, предназначенные для передачи данных и управляющей информации между процессорами.

5.6.2. Матричные системы

Основной особенностью матричных и векторных операций является то, что одна и та же команда должна выполнятся одновременно над большой совокупностью связанных между собой некоторым образом данных, т. е. реализуется *синхронный параллельный вычислительный процесс*. В подобном случае достаточно иметь одно устройство управления, которое могло бы дешифровать одиночный поток команд и вырабатывать управляющие сигналы для некоторой совокупности (матрицы, вектора) **процессорных элементов** (ПЭ) — некоторых простых (элементарных) процессоров.

Организация систем этого типа следующая: общее **управляющее устройство**, генерирующее поток команд, и большое число ПЭ, работающих параллельно и обрабатывающих каждое свой поток данных. ПЭ связываются друг с другом определенным образом, причем характер связей может быть различным, так же как и характер взаимодействия ПЭ.

Схематическое изображение конфигурации матричного процессора приведено на рис. 5.12. Отдельные процессорные элементы (элементарные процессоры) не являются законченными центральными процессорами и не могут работать самостоятельно. Единственная степень свободы ПЭ, работающего в матричной системе, состоит в том, что он может выполнять или не выполнять некоторую частную команду. Сигналы, реализующие эту возможность, вырабатываются и распределяются специальным управляющим процессором.

Первым большим достижением в этой области была разработка системы **SOLOMON** (США), за которой последовала система SOLOMON-II. Проект системы SOLOMON не был реализован, система оказалась нежизнеспособной вследствие громоздкости, недостаточной гибкости и эффективности. Однако идеи, заложенные в ней, получили развитие в системе **ILLIAC-IV**, разработанной Иллинойским университетом и изготовленной фирмой Burroughs.

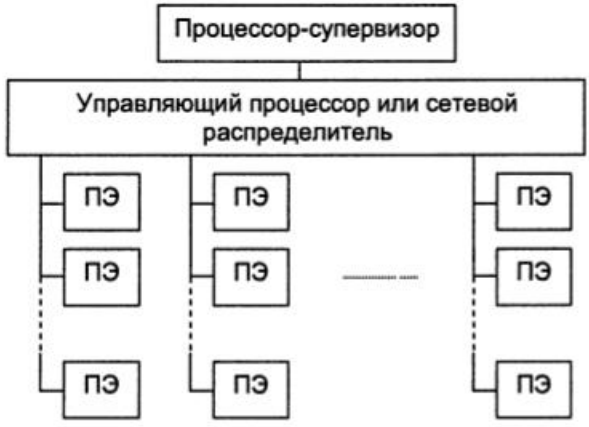


Рис. 5.12. Обобщенная схема матричной ВС

По первоначальному проекту (рис. 5.13) система ILLIAC-IV должна была включать в себя 256 ПЭ, разбитых на 4 группы — квадранты, каждый из которых должен управляться специальным процессором (УП). Кроме ПЭ и УП ВС содержала также внешнюю память и оборудование ввода-вывода. Управление всей системой предполагалось осуществлять с помощью центрального управляющего процессора (ЦУП). Однако реализовать этот замысел не удалось из-за возникших технологических трудностей при создании интегральных схем, ОЗУ и удорожания всего проекта почти в два раза. В результате с опозданием на два года (в 1971 г.) система была создана в составе одного квадранта и одного УП и с начала 1974 г. введена в эксплуатацию. Ранее предполагалось получить на этой системе производительность примерно 1 GFLOPS, однако реально было достигнуто лишь 200 MFLOPS. Тем не менее этого оказалось достаточно, чтобы система в течение ряда лет считалась самой высокопроизводительной в мире.

Рис. 5.13. Система ILLIAC-IV (проект)

В каждом квадранте (рис. 5.14) 64 ПЭ образовывали матрицу размером 8 × 8. Связь с внешней средой имели все ПЭ без исключения. Реально действовавшая система ILLIAC-IV состояла из двух частей: центральной с устройством управления и 64 ПЭ, а также подсистемы ввода–вывода, включавшей в себя универсальную ЭВМ B-6700, файловые диски и лазерную архивную память большой емкости. Каждый ПЭ состоял из 64-разрядного процессора и ОЗУ. Процессор выполнял универсальный набор операций, достигая в среднем производительности 3 MFLOPS.

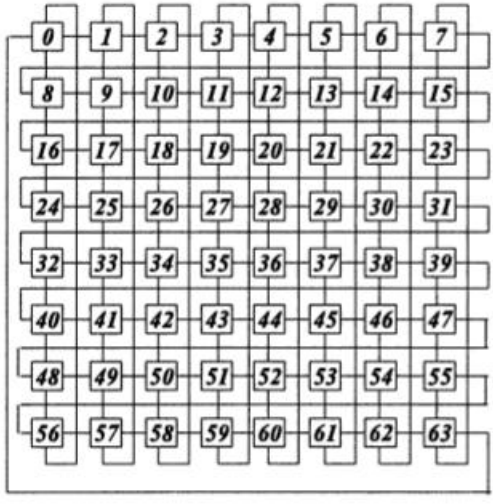


Рис. 5.14. Квадрант системы ILLIAC-IV

Емкость ОЗУ каждого ПЭ составляла 2048 64-разрядных слов при длительности цикла памяти 350 нс. Память была выполнена на интегральных схемах.

Каждый процессор имел счетчик адресов и индексный регистр. Конечный адрес мог формироваться как сумма трех составляющих: адреса, содержавшегося

в команде для данного ПЭ, кода, содержавшегося в центральном индексном регистре УУ, и кода, содержавшегося в собственном индексном регистре. Каждый процессор, кроме индексного регистра, имел в своем составе пять программно-адресуемых регистров: накапливающий сумматор, регистр для операндов, регистр пересылок (для передач информации от одного ПЭ к другому), буферный регистр на одно слово и 8-разрядный регистр управления состоянием ПЭ. В зависимости от содержимого регистра управления определялась активность или пассивность ПЭ.

Если вычисления не требовали полной разрядности, то процессор мог быть разбит на два 32-разрядных подпроцессора или даже восемь 8-разрядных. Это позволяло в случае необходимости обрабатывать векторные операнды из 64, $2 \times 64 = 128$ и $8 \times 64 = 512$ элементов.

Как видно из рис. 5.14, каждый i-й ПЭ связан с четырьмя другими: $(i - 1)$-, $(i + 1)$-, $(i + 8)$- и $(i - 8)$-м. При такой связи передача данных между любыми двумя ПЭ осуществляется не более, чем за 7 шагов, а среднее число шагов равно 4.

По шине состояния ПЭ могли передавать сигналы о состоянии в УУ. Оперативное ЗУ каждого ПЭ было связано со своим процессором, устройством управления центральной частью и подсистемой ввода-вывода, включавшей в себя универсальную ЭВМ B-6700.

Машина B-6700 выполняла и различные другие функции: трансляции программ, управления запросами на ресурсы, предварительной обработки информации и т. д.

Следует подчеркнуть, что сверхвысокая производительность системы достигалась только на определенных типах задач, таких, например, как операции над матрицами, быстрое преобразование Фурье, линейное программирование, обработка сигналов, т. е. задач с достаточно крупным зерном параллелизма. Необходимо отметить также и то, что разработка высокопроизводительных программ для системы ILLIAC-IV являлась весьма трудоемким занятием. Для упрощения этой работы были разработаны специальные алгоритмические языки.

5.6.3. Ассоциативные системы

К числу систем класса SIMD относятся **ассоциативные системы**. Эти системы, как и матричные, характеризуются большим числом операционных устройств, способных одновременно, по командам одного управляющего устройства, вести обработку нескольких потоков данных. Но эти системы существенно отличаются от матричных способами формирования потоков данных. В ассоциативных системах информация на обработку поступает от **ассоциативных запоминающих устройств (АЗУ)**.

Напомним, что АЗУ характеризуются тем, что информация из них выбирается не по определенному адресу, а по ее содержанию, задаваемому в виде признака сравнения. Запоминающие элементы АЗУ, в отличие от элементов адресуемых ЗУ,

способны не только хранить информацию, но и выполнять определенные логические функции. АЗУ позволяют осуществить поиск как по равенству содержимого ячейки заданному признаку сравнения, так и по условиям «больше (меньше)», «больше или равно (меньше или равно)».

Отмеченные выше свойства АЗУ характеризуют их преимущества для обработки информации. Формирование нескольких потоков идентичной информации с помощью АЗУ осуществляется быстро и просто, а с большим числом операционных элементов можно создавать высокопроизводительные системы. Надо учитывать еще и то, что на основе ассоциативной памяти легко реализуется изменение места и порядка расположения информации. Благодаря этому АЗУ является эффективным средством формирования наборов данных.

Целый ряд задач, таких, как обработка радиолокационной информации, распознавание образов, обработка различных снимков и других задач с матричной структурой данных, эффективно решается ассоциативными системами. При этом программирование таких задач для ассоциативных систем может быть проще, чем для традиционных ВС.

Все сказанное является предпосылкой достижения на основе ассоциативного принципа обращения к информационным словам высокой степени параллельности их обработки, когда одна операция выполняется одновременно над всем множеством содержащихся в выделенном поле ассоциативной памяти слов. При этом в простейшем случае обрабатывающее устройство — **ассоциативный процессор** — состоит из ассоциативной памяти и процессорных элементов (ПЭ), реализующих арифметическую и логическую обработку информации.

Достижение наивысшей степени производительности обработки возможно, когда число ПЭ соответствует числу обрабатываемых слов. При этом, если они обрабатываются поразрядно-последовательно, то в каждый момент времени ПЭ обрабатывают так называемый «**разрядный срез**» всех слов ассоциативной памяти. На рис. 5.15 изображена схема n-разрядной ассоциативной ВС.

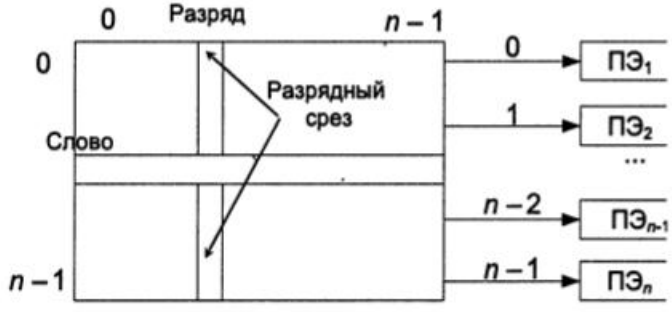


Рис. 5.15. Структура n-разрядной ассоциативной ВС

Рассмотрим более подробно структуру ассоциативной ВС на примере системы **STARAN** (США), являющейся наиболее характерным представителем систем данного класса (рис. 5.16).

Ассоциативная память в ВС STARAN имела многомерный доступ, т.е. в нее можно было обратиться как поразрядно, так и пословно. Операционные процессорные элементы были предусмотрены для каждого слова памяти. Кроме этого, система обладала уникальной схемой перестановок для перегруппировки данных в памяти.

Основным элементом системы являлась многомерная ассоциативная матрица — **ассоциативный модуль (АМ)**, который представлял собой квадрат из 256 разрядов на 256 слов, т. е. содержал 65536 бит данных. Для обработки информации предназначались 256 процессорных элементов, работавших одновременно, по одной команде, выдаваемой устройством управления. Сразу по одной команде обрабатывались все выбранные по определенным признакам из памяти слова.

Рис. 5.16. Схема ассоциативной ВС STARAN

Модуль процессорных элементов состоял из трех 256-разрядных регистров — *M*, *X* и *Y*. В состав каждого элементарного процессора входило по одному разряду из названных регистров. Регистр маски (*M*) позволял маскировать элементы 256-разрядных кодов. Регистры *X*, *Y* и связанная с ними логика были предназначены для выполнения логических операций над двумя переменными. Мультиплексор обеспечивал поочередное использование перестановочной сети с различными устройствами модуля: памятью, регистрами, магистралями ввода-вывода.

Перестановочная сеть играла важную роль при реализации операций перестроения данных в ассоциативной памяти. Она осуществляла передачу информации из одного ПЭ в другой за счет управляемой перестановки разрядов при передаче слов через сеть. Многократной передачей данных через перестановочную сеть можно было добиться сложных их перестроений как в вертикальном, так и в горизонтальном направлении.

Базовая конфигурация системы STARAN содержала один АМ. Однако число этих модулей могло варьироваться в системе от 1 до 32. При максимальной комплектации ассоциативной обработке в системе могло подвергаться 256 Кбайт информации.

Устройство управления ассоциативными модулями организовывало выполнение операций над данными по командам, хранившимся в управляющей памяти. Можно было выбирать несколько рабочих подмножеств из общего множества данных, хранимых в АМ, и выполнять над этими подсистемами операции, не затрагивая остальную информацию.

К системе подключался последовательный контроллер (однопроцессорная ЭВМ типа PDP-11). Данная ЭВМ обеспечивала работу в режиме трансляции и отладки программ, первоначальную загрузку управляющей памяти, связь между оператором и системой; управление программами обработки прерываний по ошибкам, а также программами технической диагностики обслуживания.

5.7. СИСТЕМЫ КЛАССА MIMD

5.7.1. Обобщенная структура MIMD-систем

Как уже отмечалось, к классу MIMD (МКМД) можно отнести большинство существующих многопроцессорных и многомашинных систем. Отличительной особенностью MIMD-систем является наличие в ВС нескольких обрабатывающих устройств (ОУ), на каждое из которых поступают свои потоки команд и данных. Поскольку подавляющее большинство MIMD-систем ориентированы на обработку на каждом ОУ частей одной связанной задачи, без существенного ограничения общности можно считать, что ОУ в MIMD-системе тем или иным образом связаны между собой.

Независимо от парадигмы параллельного программирования модель вычислений в MIMD-системе состоит в следующем. Каждое ОУ производит вычисления над некоторой частью общих для всей параллельной программы данных. При необходимости осуществляется информационное взаимодействие между ОУ системы с целью обмена результатами расчетов. Таким образом, в любой MIMD-системе можно выделить две основные составляющие — **вычислительную** и **коммуникационную**.

Вычислительная составляющая сосредоточивается в обрабатывающих устройствах, организованных тем или иным образом в **вычислительные модули** (**ВМ**). В большинстве случаев один вычислительный модуль ассоциируется с одним центральным процессором, хотя часто встречается и иная организация ВМ. Совокупность ВМ, непосредственно используемых для выполнения параллельных программ, часто называют **решающим полем**.

В качестве коммуникационной составляющей (рис. 5.17) выступает *коммуникационная среда*, объединяющая тем или иным образом вычислительные моду-

ли между собой. В результате можно дать следующее обобщенное определение MIMD-системы. **MIMD-система** — *это совокупность определенного числа вычислительных модулей, объединенных между собой некоторой коммуникационной средой.*

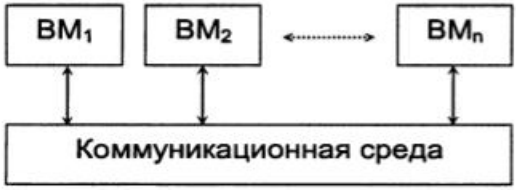


Рис. 5.17. Обобщенная структура MIMD-системы

Все MIMD-системы по степени связанности ВМ между собой разделяются на *сильносвязанные ВС* и *слабосвязанные ВС*. В **сильносвязанных** системах все ВМ имеют каким-либо образом организованную общую оперативную память и единое логическое адресное пространство. В **слабосвязанных** ВС оперативная память распределена по вычислительным модулям системы.

К вычислительным модулям MIMD-систем применимы уже знакомые показатели качества, важнейшим из которых является производительность. Коммуникационные среды имеют определенные особенности организации и свои качественно-количественные характеристики, которые подробно будут рассмотрены в следующем разделе.

Отметим, что одним из важнейших интегральных показателей качества MIMD-системы является *сбалансированность вычислительной и коммуникационной составляющих*. Фактически данный показатель определяет те классы алгоритмов и задач, которые могут быть успешно решены на той или иной MIMD-системе.

5.7.2. Коммуникационные среды MIMD-систем

5.7.2.1. Основные характеристики коммуникационных сред

Предназначением любой коммуникационной среды является передача информации от некоторого отправителя (отправителей) к некоторому получателю (получателям). Передачу информации от одного отправителя к единственному получателю называют **обменом «точка-точка»**, а к множеству получателей — **широковещательным обменом**.

Производительность обмена «точка-точка» зависит от того, является ли этот обмен единственным в коммуникационной среде, или одновременно с ним выполняются другие обмены. Не только обмены с одними и теми же участниками (например, две одновременных посылки из двух разных ВМ в один), но и обмены между разными парами ВМ могут, вообще говоря, влиять друг на друга.

При выполнении обмена (приема или передачи сообщения) вычислительным модулем процесс обмена разделяется на два элементарных акта: *запуск обмена*

и *проверка завершения*. В то время, когда обмен запущен, но еще не завершился, ВМ может выполнять некоторые расчеты. Производительность ВМ на фоне выполняющегося обмена может, вообще говоря, снижаться, поскольку идущий асинхронно обмен потребляет ресурсы ВМ: нагружает шину памяти, выставляет прерывания центральному процессору и т. п.

Производительностью коммуникационной среды для обмена «точка-точка» будем называть количество данных, передаваемых в единицу времени, в среднем за некоторый относительно большой промежуток астрономического времени. Производительность коммуникационной среды можно представить как среднюю (за длительное время) скорость передачи данных. Очевидно, производительность сильно зависит от того, какой длины сообщения используются при передаче данных, т. е. от того, какую часть времени коммуникационная среда тратит на запуск обменов.

Пусть ВМ *A* передает ВМ *B* сообщение длиной *X* байтов, и при этом никаких других обменов в среде не происходит. Время *T*, затрачиваемое на такую передачу, довольно точно оценивается формулой

$$T = \frac{X}{S} + L. \tag{5.10}$$

где *L* не зависит от *X*.

Величину *S* в этой формуле называют **пропускной способностью** *коммуникационной среды для обмена «точка-точка»*, или **мгновенной скоростью** передачи данных. Пропускная способность измеряется в байтах (мегабайтах) в секунду.

Величину *L* называют **задержкой**, или **латентностью**. **Латентность** представляет собой время запуска обмена (т. е. время, затрачиваемое коммуникационной средой на подготовку к передаче информации), не зависящее от длины сообщения, и измеряется в секундах (микросекундах). Очень часто латентность определяют как *время передачи сообщения нулевой длины*.

Главными факторами, влияющими на величину латентности, являются:

- число копирований в промежуточные буферы;
- быстродействие протокола установления соединения;
- алгоритм маршрутизации в составных коммутаторах;
- задержки на физическом уровне, связанные с быстротой срабатывания электронных схем.

Иногда удобно оперировать латентностью, приведенной к пропускной способности, или **ценой обмена**, которую мы обозначим как *P*:

$$P = L \cdot S. \tag{5.11}$$

Эта величина измеряется в байтах и может быть интерпретирована следующим образом. Цена обмена представляет собой то число байт, которое коммуникационная среда могла бы передать при обмене «точка-точка» за время своего запуска, т. е. за время задержки. Иными словами, за счет инертности коммуника-

ционной среды к каждому передаваемому ей сообщению «добавляется» с точки зрения пропускной способности P байт.

Таким образом, производительность коммуникационной среды зависит от длин передаваемых сообщений. Если X много больше P, т. е. длины сообщений много больше цены обмена, производительность близка к пропускной способности. Напротив, если X много меньше P, производительность практически полностью определяется латентностью, а не пропускной способностью. Наконец, при X, равном P, производительность равна в точности половине пропускной способности. Тем самым, *цена обмена — это такая длина сообщения, при использовании которой производительность коммуникационной среды при обмене «точка-точка» равна половине ее пропускной способности.*

Очевидно, коммуникационная среда тем лучше, чем выше пропускная способность и чем ниже латентность.

Следующей важной характеристикой коммуникационной среды является ее **полнота**. Содержательно полнота среды есть мера того, насколько одновременно происходящие обмены влияют друг на друга, снижая производительность. Простейшее определение полноты коммуникационной среды — это понятие **бисекционной полноты**: среда называется **бисекционно полной**, если любые обмены между разными парами ВМ, происходящие одновременно и в любом количестве, не снижают производительности друг друга. Другими словами, при любом разделении коммуникационной среды пополам (бисекции) пропускная способность потока данных из одной половины в другую есть сумма пропускных способностей независимых каналов связи, ведущих из одной половины среды в другую. Типичный пример **бисекционно неполной** коммуникационной среды — сеть на базе двух однородных коммутаторов, объединенных единственным каналом связи.

5.7.2.2. Простые коммутаторы

В настоящее время в мировой практике построения ВС существует большое разнообразие коммуникационных сред. При этом постоянно ведутся теоретические и экспериментальные исследования в области построения моделей коммуникационных сред, обладающих максимальной пропускной способностью и минимальной задержкой. Любая коммуникационная среда объединяет совокупность ВМ с возможностью передачи информации от одного ВМ к другому, т. е., другими словами, осуществляет функцию коммутации ВМ между собой. Поэтому в дальнейшем, в целях унификации рассмотрения различных коммуникационных сред, будем говорить о них как о коммутаторах.

Коммутаторы бывают **простые** и **составные**, компонуемые из простых. Простые коммутаторы имеют малую задержку при установлении полнодоступных соединений, но в силу физических ограничений могут быть построены только для систем с небольшим числом вычислительных модулей. Для построения коммутаторов с большим числом входов и выходов развита теория составных коммутаторов, строящихся из простых путем объединения их в многокаскадные схемы

линиями «точка-точка». Способы такого объединения разнообразны и зависят от требований, предъявляемых к коммутатору как элементу вычислительной системы.

Простые коммутаторы бывают с *временным* и *пространственным* разделением. Простой коммутатор с временным разделением, называемый также *шиной*, используется, например, в SMP-серверах Power Challenge фирмы Silicon Graphics Inc. Простой коммутатор с пространственным разделением Gigaplane применяется фирмой Sun в семействе SMP-систем Sun Ultra Enterprise.

Организация шин подробно рассматривалась в разделе, посвященном вводу-выводу в ЭВМ, поэтому здесь мы ограничимся напоминанием основных достоинств и недостатков шинных структур.

К достоинствам шин следует отнести простоту реализации и малое количество аппаратного оборудования, необходимого для реализации. Кроме этого, шинные структуры в силу простоты отличаются высокой надежностью. Главным недостатком шинной организации является то, что шина представляет собой критический ресурс, за доступ к которому конкурируют абоненты. Неизбежные конфликты приводят к снижению производительности шины (прежде всего, увеличиваются задержки), она становится узким местом системы. В результате число шинных абонентов является ограниченным (при том весьма небольшим числом), что существенно сдерживает масштабирование ВС.

Отметим, что локальные сети Ethernet и Token Ring с протоколами, использующими общую среду распространения сигналов, также служат коммутаторами с временным разделением и реализуют логический протокол шинной структуры.

Коммутаторы с пространственным разделением способны соединить любой вход с любым одним выходом (**ординарные**) или несколькими выходами (**неординарные**). Каждое соединение входов и выходов выполняется разными частями аппаратуры, что и дало название этому типу коммутаторов.

Коммутатор (рис. 5.18) представляет собой совокупность мультиплексоров, число которых равно числу выходов коммутатора. Количество входов коммутатора может быть произвольным. Каждый вход должен быть заведен на все мультиплексоры коммутатора.

Достоинствами коммутаторов с пространственным разделением являются полнодоступность и минимальная латентность. К недостаткам следует отнести высокую сложность при большом числе входов и выходов, приводящую, как следствие, к существенному снижению надежности. Кроме этого, существуют ограничения по количеству выходов коммутатора, связанные с возможностями размножения входов для подсоединения к каждому мультиплексору.

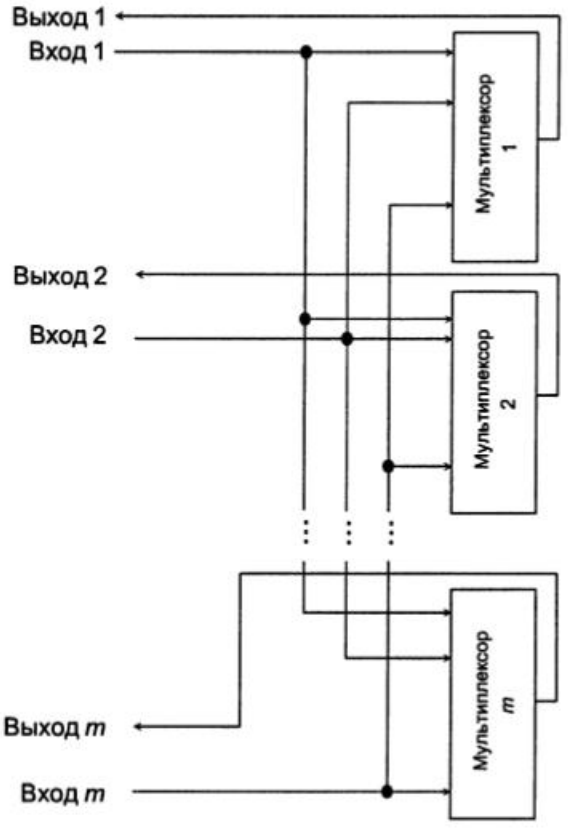


Рис. 5.18. Простой коммутатор $m \times m$ с пространственным разделением

5.7.2.3. Составные коммутаторы

Простые коммутаторы как с временным, так и пространственным разделением имеют ограничения по числу входов и выходов, а коммутаторы с пространственным разделением, кроме того, требуют для своего построения большого объема оборудования.

Коммутаторы с большим числом входов и выходов можно построить из простых коммутаторов с меньшим числом входов и выходов путем объединения простых коммутаторов линиями «точка-точка».

Объем оборудования составного коммутатора значительно меньше, чем у простого коммутатора с теми же количествами входов и выходов. Однако составные коммутаторы имеют задержку, как минимум, пропорциональную числу **каскадов** — числу простых коммутаторов, через которые передаваемая информация проходит от входа до выхода составного коммутатора.

Как правило, составные коммутаторы строятся из простых коммутаторов $n \times n$ с n входами и n выходами. Если используется коммутатор с временным раз-

делением, то коммутации входов с соответствующими выходами разносятся во времени.

На рис. 5.19 представлена схема простого коммутатора, который можно использовать в качестве элемента для построения составного коммутатора. Тонкие линии представляют управляющие сигналы, двойные линии — входящие и выходящие информационные сигналы. Функционирование блока коммутации данных определяется значением управляющего сигнала блока управления, задающего, какой вход с каким выходом необходимо соединить.

Рис. 5.19. Элемент составного коммутатора

Блок управления вырабатывает управляющий сигнал и производит арбитраж запросов двух и более входов, если они желают соединиться с одним выходом.

Запрос r_i воспринимается на соответствующем входе блока управления, если есть сигнал 1 на соответствующей *линии запроса i*. При наличии значения 1 на *линии запроса i*, $i \in \{0, n-1\}$ вход i блока коммутации соединяется с выходом, номер которого задается управляющим числом d_i на входе *управление i*.

В случае конфликта, когда есть сигнал 1 на *линии запроса i* и на *линии запроса* j, $d_i = d_j$, $i < j$, предпочтение отдается запросу r_i, и вырабатывается сигнал занятости b_j. Сигнал занятости передается коммутатору предыдущего каскада.

При отсутствии конфликта или его разрешении в пользу r_i блок управления коммутирует сигнал *линии запроса i*, $i \in \{0, n-1\}$ в соответствии с управляющим числом d_i так, чтобы сигнал запроса R_{d_i} был равен 1.

Важным моментом функционирования блока управления является распространение сигнала занятости. Если сигнал на входе B_i, $i \in \{0, n-1\}$, равен 1, что происходит при конфликте в коммутаторе следующего каскада, то этот сигнал распространяется на выход j, для которого верно соотношение $d_j = i$.

Отметим, что простой коммутатор $n \times n$ может иметь или не иметь внутренней буферизации.

Составной коммутатор создается путем объединения простых коммутаторов $m \times n$, входы которых либо являются входами составного коммутатора, либо подключены к выходам одного или разных других простых коммутаторов. Оставшиеся при таком объединении свободными выходы простых коммутаторов служат выходами составного коммутатора. Структура такого составного коммутатора задается ориентированным графом, каждая вершина которого имеет m входящих ребер и n выходящих.

Установление соединения входов с выходами называется **коммутацией**. Для установления коммутации требуется подать управляющие числа d_i на каждый вход каждого из простых коммутаторов, образующих составной. Последовательность управляющих чисел d_i для простых коммутаторов каждого каскада будет образовывать **маршрут** соединения. Задача определения маршрутов возлагается на управляющий блок составного коммутатора.

Составной коммутатор называется **полнодоступным**, если может быть установлено соединение любого входа с любым выходом. В противном случае, если хотя бы один вход не может быть соединен по крайней мере с одним выходом, то коммутатор является **неполнодоступным**.

Коммутатор называется **неблокируемым**, если может быть установлено соединение свободных входа и выхода при наличии произвольного количества установленных соединений других входов и выходов. Если хотя бы одно такое соединение не может быть установлено, коммутатор называется **блокируемым**. Если для вновь соединяемых входа и выхода может быть установлено соединение путем перекоммутации уже существующих соединений с их сохранением, то такой коммутатор называется **неблокируемым с перекоммутацией**.

Составные коммутаторы с числом входов N и выходов M могут быть построены с использованием различного числа по-разному соединенных друг с другом простых коммутаторов. Используемые простые коммутаторы могут иметь различное число входов и выходов.

Критериями эффективности при построении составных коммутаторов являются следующие:

- количество используемых простых коммутаторов (чем меньше, тем лучше);
- количество коммутаторов, участвующих в соединении любого входа с любым выходом (число каскадов);
- сложность алгоритма определения маршрутов и установления коммутации.

Далее рассмотрим в качестве иллюстрации два часто встречающихся типа составных коммутаторов — коммутатор Клоза и баньян-сети.

5.7.2.4. Коммутатор Клоза

Коммутатор Клоза является составным трехкаскадным коммутатором с $m \times d$ входами и $m \times d$ выходами. Он формируется из m входных коммутаторов $d \times d$, m выходных коммутаторов $d \times d$ и d промежуточных коммутаторов $m \times m$.

Правила соединения простых коммутаторов следующие (рис. 5.20):

- k-й выход i-го входного коммутатора соединяется с i-м входом k-го промежуточного;
- k-й вход j-го выходного коммутатора соединяется с j-м выходом k-го промежуточного.

Коммутаторы Клоза появились в 1953 г., регулярный алгоритм соединений был разработан в 1969 г. Коммутатор способен соединять любой вход с любым выходом, однако при уже установленных соединениях части входов и выходов добавление соединения еще одного входа и выхода может потребовать переустановления всех соединений.

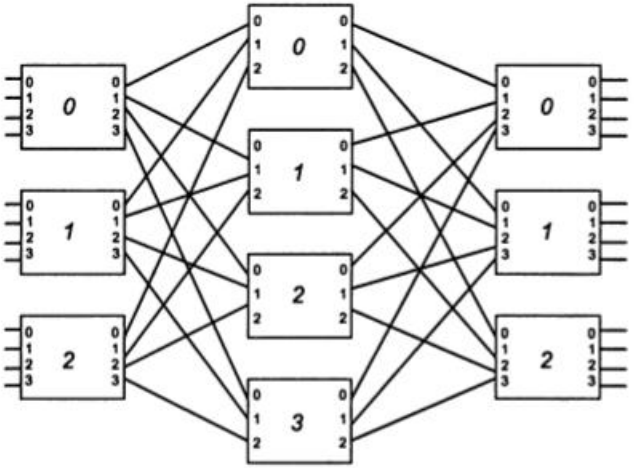


Рис. 5.20. Коммутатор Клоза

Таким образом, коммутатор Клоза является полнодоступным неблокируемым с перекоммутацией ординарным коммутатором. Достоинством схемы Клоза является простота и масштабируемость, достигаемые при сравнительно небольшом числе простых коммутаторов. Главным недостатком следует признать то, что коммутатор Клоза в некоторых случаях требует перекоммутации. На практике возможность перекоммутации часто не может быть реализована без потери передаваемых данных, что фактически делает коммутатор Клоза блокируемым.

5.7.2.5. Баньян-сети

При конструировании **баньян-сетей** преследовалась цель максимально упростить алгоритмы определения маршрутов и установления коммутации. Баньян-сети строятся на базе простых коммутаторов $a \times b$ таким образом, что существует только один путь от каждого входа к каждому выходу. Рассмотрим баньян-сети на примере их важного подкласса — так называемых **дельта-сетей**, введенных Пателом в 1981 г.

Дельта-сеть представляет собой n-каскадный коммутатор с a^n входами и b^n выходами. Основой построения дельта-сети служат простые коммутаторы $a \times b$ (рис. 5.21).

Обозначим чрез S_{qc} *q-тасовку q·c* объектов.

$$S_{qc}(i) = q \cdot i \bmod(q \cdot c - 1 \quad \text{для } 0 \le i < q \cdot c - 1;$$
$$S_{qc}(i) = i \text{ для } i = q \cdot c - 1. \tag{5.12}$$

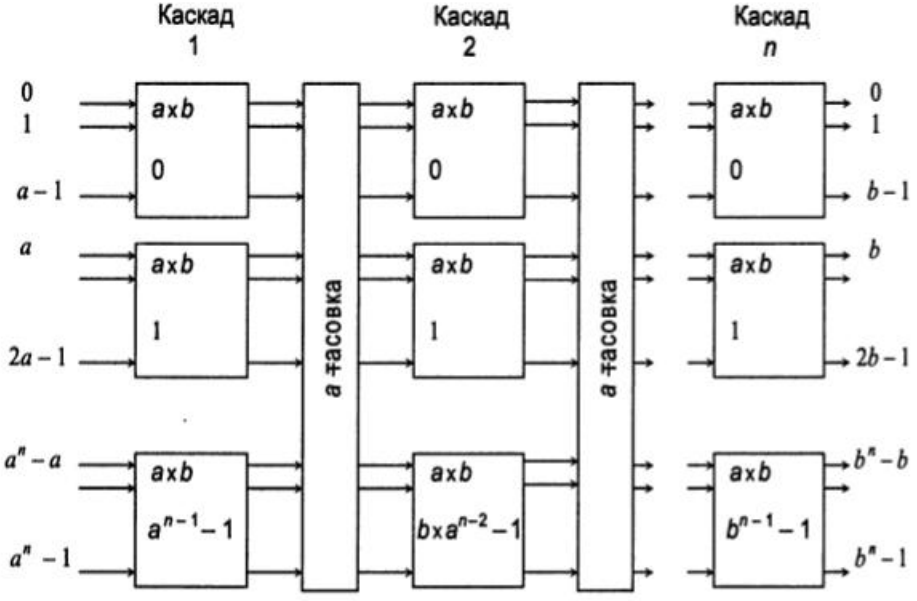


Рис. 5.21. Схема построения дельта-сети

При построении $a^n \times b^n$ дельта-сети межкаскадные соединения выполняются в соответствии с a-тасовкой. Тасуемыми объектами являются выходы коммутаторов очередного каскада. Каскады нумеруются числами 1, 2, ..., n. Первый каскад состоит из a^{n-1} коммутаторов и имеет $b \times a^{n-1}$ выходов, что равно соответственно числу входов второго каскада. Второй каскад состоит из $b \times a^{n-2}$ коммутаторов, i-й каскад состоит из $b^{i-1} \times a^{n-i}$ коммутаторов.

На рис. 5.22 представлен пример дельта-сети $2^3 \times 2^3$, построенной на основе коммутаторов 2×2.

Показатель степени числа входов определяет число каскадов (в нашем случае — 3 каскада). Числа a и b равны 2 — основанию степени числа входов/выходов. Число q в формуле q-тасовки, поскольку в дельта-сети используется a-тасовка, определяется значением a (2). Между каскадами необходимо соединить 8 выходов предыдущего каскада с 8 входами следующего, т. е. тасуется 8 объектов. Отсюда определяется число c в формуле q-тасовки (тасуется $q \cdot c$ объектов) — оно равно $\frac{q \cdot c}{q} = \frac{8}{2} = 4$.

Выходы будут соединяться со входами согласно значениям функции S_{24}:

$S_{24}(0) = 0$;

$S_{24}(1) = 2$;

$S_{24}(2) = 4$ и т. д.

Дельта-сеть является полнодоступным неблокируемым ординарным коммутатором. Неблокируемость дельта-сети и простота установления коммутации (нетрудно вычислить единственный существующий маршрут между входом и выходом) являются достоинствами приведенной схемы.

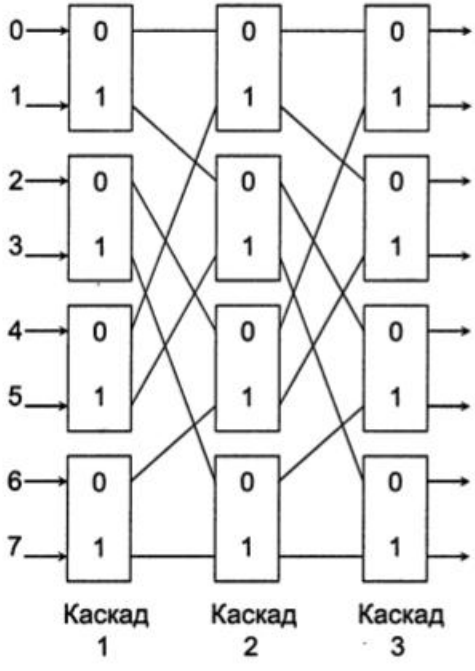


Рис. 5.22. Дельта-сеть 8×8

К недостаткам следует отнести:

• большее по сравнению со схемой Клоза число простых коммутаторов при том же числе входов и выходов составного коммутатора;

• для обеспечения большого числа входов и выходов дельта-сети требуется построение многокаскадного коммутатора (поскольку число входов и выходов простых коммутаторов ограничено), что увеличивает задержки.

5.7.2.6. Распределенные составные коммутаторы

Рассмотренные выше коммутаторы представляют собой законченные устройства. Альтернативный способ построения составных коммутаторов заключается в следующем. Составной коммутатор строится из коммутаторов с $v + 1$ входом и $v + 1$ выходом, $v \geq 2$. При этом один вход каждого составляющего коммутатора служат входом и выходом составного коммутатора. Полагается, что к каждому простому коммутатору подключаются процессор и память, образуя вместе с составляющим коммутатором вычислительный модуль с v каналами для соединения с себе подобными.

Свободные v входов и v выходов каждого ВМ соединяются линиями «точка-точка» с входами и выходами других ВМ, образуя **граф межмодульных связей**. Построенные таким образом составные коммутаторы называются **распределенными**, в отличие от рассмотренных ранее, которые называются **сосредоточенными**.

С точки зрения простоты организации передач данных между вычислительными модулями наиболее подходящий граф межмодульных связей — полный. При полном графе каждая пара ВМ имеет хотя бы одну линию связи. Однако в большинстве случаев связать каждую пару ВМ отдельным каналом связи не представляется возможным, что порождает проблему выбора графа межмодульных связей.

Отсутствие линий связи между ВМ приводит к тому, что обмен данными между ними выполняется через цепочки транзитных ВМ. Это требует определения трасс передачи данных, что является трудоемкой задачей теории графов. Очевидно, наличие транзитных передач отрицательно сказывается на времени передачи информации. Кроме этого, ограничивается число информационных обменов, которые могут проходить одновременно. Граф межмодульных связей должен выбираться исходя из минимизации времени выполнения межмодульных обменов и максимизации числа одновременно выполняемых обменов.

При выборе графа межмодульных связей важную роль играет учет отказов и восстановлений ВМ и линий связи. В каждый момент времени совокупность исправных ВМ и линий определяет существующий граф межмодульных связей. Этот граф может быть как связным, так и несвязным, состоящим из нескольких

компонент связности. Так как решение параллельной задачи подразумевает обмен данными, то выбор графа должен производиться исходя из максимального сохранения связности при выходах из строя ВМ и линий.

Количество необходимых транзитных передач определяется расстоянием между вершинами графа межмодульных связей, соответствующих передающему и принимающему ВМ. Под **расстоянием** между вершинами графа понимается количество ребер кратчайшего пути между этими вершинами. Расстояние между максимально удаленными вершинами называется **диаметром** графа. Диаметр графа определяет максимально необходимое число транзитных передач.

В качестве показателя, оценивающего среднюю длину транзитного пути, используется **средний диаметр** графа, под которым понимается математическое ожидание расстояния между вершинами при равновероятном выборе пар вершин.

Очевидно, что минимизация диаметра и среднего диаметра графа межмодульных связей ведет к сокращению числа транзитных передач и, следовательно, к увеличению производительности коммуникационной среды.

Заметим, что все основные свойства графа межмодульных связей определяются его **топологией**. Варианты наиболее часто используемых топологий представлены на рис. 5.23.

Кроме перечисленных, важной характеристикой графа межмодульных связей является его **вектор-функция структурной живучести**. Вектор-функция структурной живучести графа представляет собой множество $\{p_1, p_2, \ldots, p_i, \ldots, p_N\}$, где N — число вершин графа.

При заданных значениях коэффициентов готовности ВМ и линий связи координата i вектор-функции задает вероятность p_i существования в графе межмодульных связей компоненты связности, включающей не менее i вершин.

Для вычисления координат вектор-функции структурной живучести требуется проведение статистических экспериментов. Ряд экспериментальных исследований показал, что минимизация диаметра и среднего диаметра является необходимым условием максимизации координат вектор-функции структурной живучести.

Таким образом, выбор графа межмодульных связей для ВС с заданными числом ВМ N и числом линий связи каждого ВМ ν может быть сведен к построению графа из N вершин со степенями ν, имеющего минимальные диаметр и средний диаметр. Однако построение такого графа является сложной задачей для большого числа вершин. Поэтому на практике часто используются графы межмодульных связей, конструируемые на основе гиперкубов («линейка», «решетка», «куб») в булевом и евклидовом пространствах.

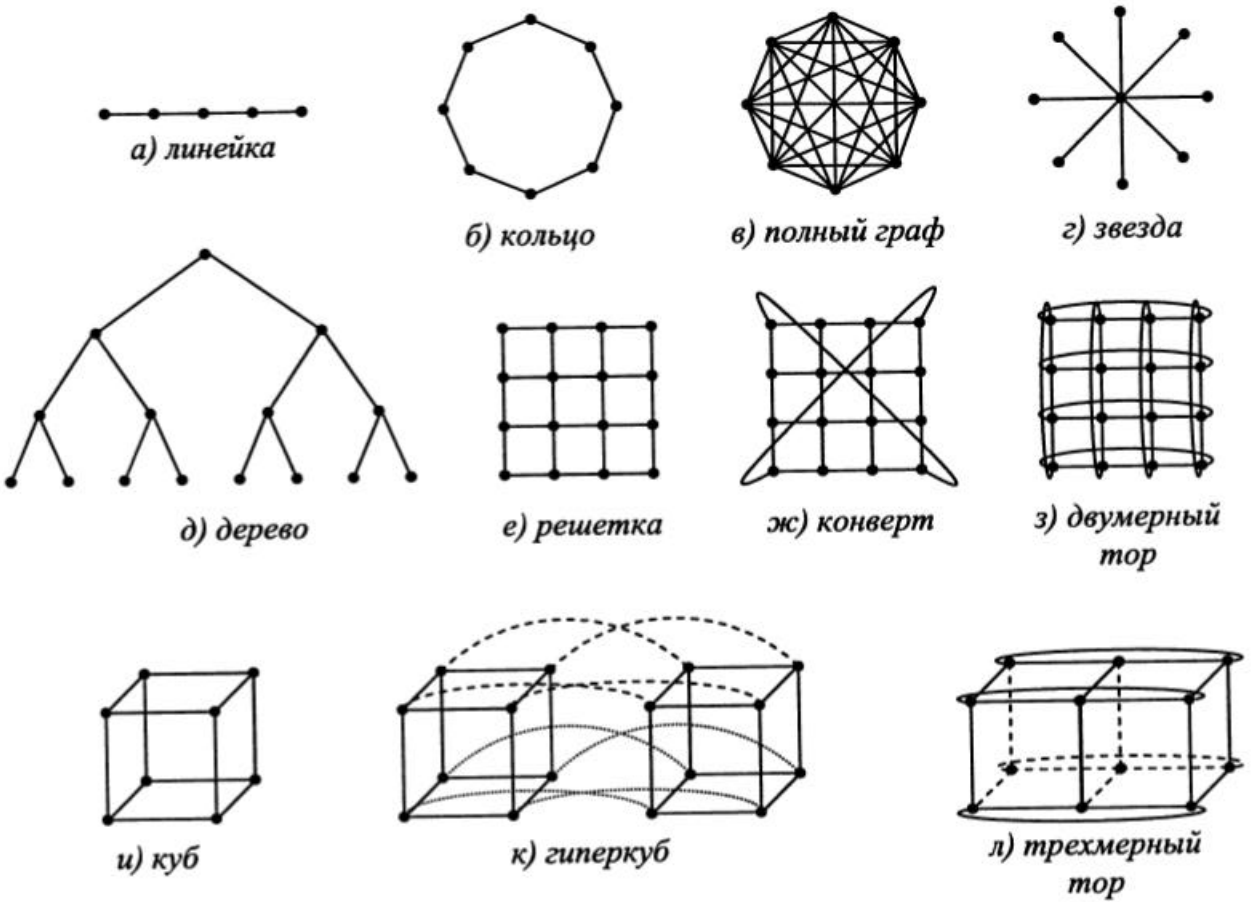


Рис. 5.23. **Различные топологии графа межмодульных связей**

5.7.2.7. Управление коммутаторами

Для передачи данных может использоваться метод *коммутации каналов* или *коммутации пакетов*.

При **коммутации каналов (circuit switching)** перед передачей данных устанавливается соединение между отправителем и получателем. Сами данные передаются только по установленному соединению. При этом может быть передан произвольный объем данных. После завершения передачи данных соединение разрушается.

При **коммутации пакетов (store-and-forward switching)** передаваемые данные оформляются в виде **пакетов**, содержащих, как правило, **заголовок (head)**, сами **данные** и **концевик (tail)**. Для удобства буферизации пакеты имеют один и тот же размер, передаваемые данные либо нарезаются на множество пакетов, либо целиком помещаются в один пакет.

Коммутация пакетов требует полного приема пакета во входной буфер коммутатора. Предполагается, что каждый вход каждого составляющего простого коммутатора имеет очередь буферов.

После приема пакета в очередь на основании маршрутной информации, содержащейся в заголовке пакета, алгоритм маршрутизации определяет выход коммутатора, через который должен быть передан данный пакет. Если требуемый выход простого коммутатора занят, пакет ожидает в буферной очереди. И так вплоть до достижения требуемого выхода составного коммутатора.

Временное хранение пакетов в буферных очередях, с одной стороны, значительно облегчает поиск маршрутов и увеличивает гибкость алгоритма маршрутизации. С другой стороны, многократные запоминания в промежуточных коммутаторах могут значительно увеличить время передачи пакета.

При коммутации каналов передаваемые данные сообщения также могут оформляться в виде пакетов. Для установления маршрута используется заголовок сообщения, что позволяет избежать приема всего пакета во входной буфер коммутатора. Заголовок передается от входа к выходу составного коммутатора, прокладывая маршрут для данных. Маршрут остается занятым на время передачи данных (пакета данных), что может привести к временным блокировкам других обменов. С другой стороны, после установления маршрута передача данных осуществляется без задержек.

Прокладывание маршрутов во время передачи заголовка и закрепление каналов на время передачи пакетов аналогичны ходам, которые проделывает червь. По этой причине указанный способ передачи данных получил название «**wormhole routing**» — «**червячная маршрутизация**».

Порядок соединения входа составного коммутатора с требуемым выходом (составление маршрута) определяется тем или иным **алгоритмом маршрутизации**, который выполняется блоком управления коммутатора. Блок управления коммутатора для каждого поступающего пакета данных формирует последовательность

управляющих чисел d_1, d_2,..., d_N, где N — число каскадов. Управляющие числа подаются на входы соответствующих простых коммутаторов (рис. 5.19).

Основной задачей алгоритмов маршрутизации является нахождение по возможности кратчайшего пути между отправителем и получателем данных.

Алгоритмы маршрутизации могут быть *детерминированными*, *адаптивными* и *частично адаптивными*. **Детерминированные** алгоритмы, используя известный граф связей составного коммутатора, устанавливают статическую систему маршрутов, которая не может быть изменена во время работы коммутатора. При всей простоте детерминированной схемы маршрутизации ее применение сильно затрудняется сбоями и отказами простых коммутаторов и линий связи. Кроме этого, новый маршрут часто прокладывается при ранее установленных соединениях, т. е. некоторые каналы могут быть занятыми.

Адаптивные алгоритмы способны учитывать занятость и отказы простых коммутаторов и линий связи, но, как правило, имеют высокую сложность. **Частично адаптивные** алгоритмы функционируют как детерминированные при малой интенсивности потоков данных и отсутствия конфликтов. При обнаружении конфликтов для их разрешения осуществляется переход на адаптивную схему маршрутизации.

Любой алгоритм маршрутизации реализуется как *система процессов, манипулирующих с буферами для приема и передачи данных* (***буферным пулом***). Поскольку число буферов ограничено, может возникнуть проблема блокировок – так называемых **дедлоков** (deadlock). Проиллюстрируем данную проблему на следующем примере.

Пусть имеется общий буферный пул. Когда алгоритм маршрутизации нуждается в буфере, он запрашивает любой буфер из пула. Если свободного буфера нет, алгоритм останавливается в ожидании буфера.

Дедлок при таком алгоритме маршрутизации возникнет, если граф межмодульных связей имеет хотя бы один цикл. В самом деле, пусть в каждом простом коммутаторе, входящем в некоторый цикл графа, буферный пул заполнен сообщениями, адресованными следующему за этим коммутатором в цикле. Возникла ситуация, когда все коммутаторы цикла, не имея свободного буфера для приема сообщения, вошли в состояние бесконечного ожидания, т. е. образовался дедлок.

При коммутации каналов для предотвращения дедлоков следует избегать циклических топологий составного коммутатора и схем маршрутизации, допускающих циклические маршруты.

При коммутации пакетов для борьбы с дедлоками используется техника **структурирования буферного пула**. Рассмотрим два основных способа такого структурирования.

Буферный пул может структурироваться **по линиям связей**. Каждый буфер ассоциируется с определенной линией связи, при этом в буфере могут размещать-

ся только те пакеты, которые предназначены для передачи по ассоциированной с буфером линией связи.

Для составления бездедлоковой схемы маршрутизации производятся следующие преобразования:

1) каждое неориентированное (т. е. двунаправленное) ребро графа межмодульных связей заменяется двумя противоположно ориентированными ребрами;

2) производится разбиение графа на множество ориентированных подграфов, при этом каждое ребро может принадлежать одному и только одному подграфу, а вершина может входить в произвольное число подграфов.

При составлении маршрутов фиксируется порядок прохождения подграфов — составляется **граф зависимости**. Граф зависимости не должен содержать циклов, т. е. при маршрутизации пакета не должны допускаться повторные заходы в пройденные подграфы.

Если существует разбиение графа межмодульных связей на ациклические ориентированные подграфы, и при любом маршруте любая пара подграфов проходится в одном и том же порядке, то маршрутизация будет бездедлоковой. Данная схема маршрутизации получила название **маршрутизации с ациклическим графом зависимости**.

Основным недостатком описанного способа является то, что маршрутизация с ациклическим графом зависимости не всегда позволяет использовать кратчайшие маршруты передачи данных. Например, известную топологию «конверт» (рис. 5.23, *ж*) можно разложить на ациклические подграфы так, как показано на рис. 5.24.

Для любого возможного маршрута фиксируем порядок прохождения — сначала подграф 1, затем — подграф 2. На рис. 5.24 отмечены две точки, минимальное расстояние между которыми равно 2, но из-за зафиксированного порядка прохождения подграфов реальное расстояние будет равно 6.

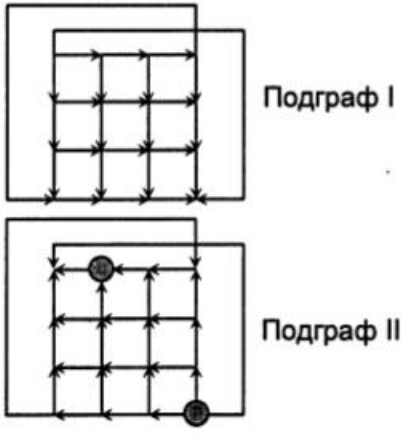


Рис. 5.24. Разложение графа с топологией «конверт» на ациклические подграфы

Простейшей реализацией способа структурирования буферного пула по линиям связи является алгоритм **«вверх-вниз»** (up-down). Из всех вершин графа межмодульных связей выбирается одна, в дальнейшем называемая *корневой*. Из корневой вершины строится нисходящее покрывающее дерево, включающее все вершины графа, а также обратное ему восходящее покрывающее дерево. Заметим, что дерево — принципиально ациклическая топология.

Алгоритм маршрутизации «вверх-вниз» заключается в следующем. Пакет данных от отправителя сначала направляется по восходящему дереву к корневой вершине. Если получатель не находится на пути следования по восходящему дереву, пакет достигает корневой вершины. Далее пакет направляется по нисходящему дереву от корневой вершины к получателю. Маршрутизация по данному алгоритму является бездедлоковой, однако почти все маршруты имеют длину, превосходящую минимально необходимую.

Другим способом структуризации буферного пула является **структуризация по расстояниям** (или **по числу пройденных перегонов**). В данном способе буферный пул является общим для всех пакетов, но при этом буферы разбиваются на классы, по крайней мере, с одним буфером в каждом классе. Число классов равно диаметру графа плюс 1.

Пакеты помечаются числом *транзитных передач* (*пройденных перегонов*), которым они подверглись. При порождении пакета, он помечается нулем. Значение отметки задает класс пакета. Пакет класса n может размещаться только в буферах классов от 0 до n, но не выше. При этом каждый пакет будет ожидать только в том случае, когда буфер занят пакетом, находящимся ближе к своему получателю, чем рассматриваемый. Поскольку маршруты конечны, то циклическое ожидание невозможно, следовательно, данный способ маршрутизации — бездедлоковый.

Несмотря на то что структуризация буферного пула по расстояниям позволяет использовать кратчайшие маршруты между отправителем и получателем, указанный способ обладает существенным недостатком, снижающим производительность составного коммутатора. Дело в том, что указанный алгоритм требует наличия в передающем промежуточном коммутаторе информации о состоянии буферного пула в принимающем коммутаторе. Промежуточные коммутаторы обмениваются информацией о состоянии буферного пула путем передачи служебных пакетов «запрос-ответ», что, очевидно, создает дополнительный управляющий трафик и увеличивает задержки в коммуникационной среде.

5.7.3. MIMD-системы с общей памятью

5.7.3.1. Архитектура MIMD-систем с общей памятью

Как уже было отмечено при классификации параллельных ВС, в составе класса MIMD выделяют подкласс **сильносвязанных** ВС, в которых связь вычислительных модулей между собой осуществляется через общее адресное про-

странство памяти. Поэтому данные ВС называются системами с **общей** или **разделяемой** памятью.

Заметим, что общим для всех ВМ системы является логическое адресное пространство памяти. При физической реализации разделяемой памяти различают два основных подхода.

Первый подход подразумевает, что общая память физически сосредоточена в одном месте, и все ВМ системы имеют к ней одинаковый (равноправный, **симметричный**) доступ через некоторую коммуникационную среду. Данная архитектура получила название **«симметричный мультипроцессор» (Symmetrical MultiProcessor — SMP)**, или архитектуры *с* **сосредоточенной общей памятью**. Каждый ВМ в SMP-системе имеет свою кэш-память, используемую для ускорения доступа к основной памяти (рис. 5.25).

В качестве коммуникационной среды в SMP-системах используются простые коммутаторы как с пространственным, так с временным разделением. Как уже было сказано, примером реализации SMP с коммутатором с пространственным разделением служит семейство Sun Ultra Enterprise. Примером шинной организации коммуникационной среды являются системы SGI Power Challenge.

К классу SMP относятся все многопроцессорные серверы, многие мэйнфреймы и некоторая часть супер-ЭВМ. Основной проблемой при построении SMP является организация общего доступа к основной памяти большого числа процессоров.

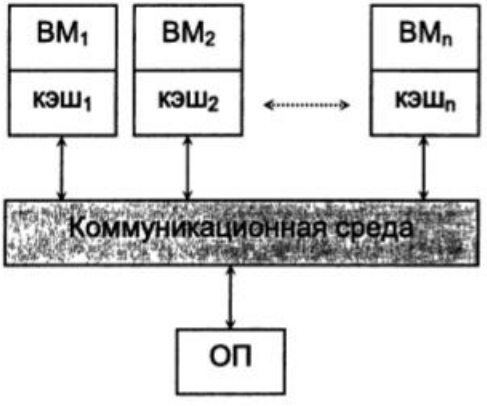


Рис. 5.25. Архитектура SMP

Большинство мировых производителей системных плат достаточно уверенно освоили массовый выпуск 2-процессорных SMP-систем, на базе которых строятся серверы до уровня отдела. Производство 4-процессорных системных плат освоено лишь наиболее известными фирмами (ASUS, Compaq), поскольку инженерно-технологические трудности построения SMP-систем велики уже на этом уровне масштабирования.

Технологией изготовления SMP-систем с 8 и более (максимум — 64) процессорами в мире владеют лишь несколько уникальных инженерных коллективов, содержать которые могут позволить себе только самые известные компании — Intel, IBM, SGI, Sun, HP, NEC, Hitachi и Fujitsu. Число производителей мощных многопроцессорных SMP-систем не превышает числа производителей универсальных микропроцессоров. Естественно, получающиеся на выходе системы отличаются высокой ценой.

Несмотря на высокую стоимость, SMP-серверы находят своего потребителя, поскольку общая память — наиболее производительная коммуникационная среда из всех существующих, имеющая максимальную пропускную способность и минимальную латентность. Именно поэтому класс решаемых на подобных ВС задач практически неограничен. На SMP-системах с успехом могут быть реализованы все существующие парадигмы и модели параллельного программирования. Наиболее широко распространенными тем не менее являются модели «параллелизм данных» и «общая память».

В то же время сравнительно небольшое число ВМ (до 64) в SMP-системе ограничивает ее общую пиковую производительность, что затрудняет построение на базе SMP систем класса супер-ЭВМ. Для преодоления указанного ограничения применяется второй подход — построение систем с **распределенной общей памятью**.

В такой системе каждый ВМ имеет собственную оперативную память, составляющую отдельный участок в общем адресном пространстве системы. При обращении в память аппаратура определяет, в какой ВМ должен быть адресован запрос на чтение или запись. Поскольку обращение к собственной памяти производится намного быстрее, чем обращение к памяти других ВМ, подобная архитектура получила название архитектуры с **неоднородным доступом к памяти** (Non-Uniform Memory Access — NUMA) (рис. 5.26).

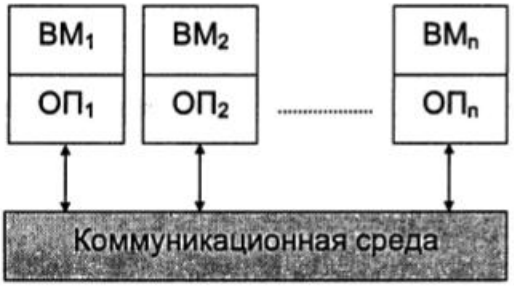


Рис. 5.26. Архитектура NUMA

NUMA-системы способны сосредоточивать внушительную вычислительную мощь. Например, NUMA-системы фирмы SGI (Altix, Origin) содержат несколько сотен процессоров.

5.7.3.2. Обеспечение когерентности памяти в ВС с общей памятью

При организации совместной работы нескольких ВМ в адресном пространстве общей памяти возникает проблема обеспечения **когерентности**, или **одинаковости**, всех копий одного и того же физического блока памяти. Копии могут содержаться в основной памяти, в кэшах и памятях различных ВМ. Очевидно, что для нормального протекания вычислительного процесса при изменении одной из копий должны быть изменены и другие. Обычно блок физической памяти и его копии называются строкой памяти.

Для обеспечения когерентности используется специальный аппаратный механизм, называемый **протоколом когерентности**. Существуют два класса таких протоколов.

1. Протоколы на основе справочника (directory based). Информация о состоянии строки памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по различным ВМ системы).

2. Протоколы наблюдения (snooping). Данные протоколы применимы к ВС, использующим для передачи данных широковещательный коммутатор (когда один ВМ передает данные, остальные имеют возможность их читать), например, общую шину. Каждый кэш, который содержит копию некоторой строки памяти, имеет также соответствующую служебную информацию о ее состоянии. Во время операций чтения и записи строк памяти кэш-контроллеры всех ВМ наблюдают за общей шиной (просматривают ее). Во время наблюдения кэш-контроллер определяет, не содержит ли его кэш копию читаемой (записываемой) строки памяти.

Имеются два подхода поддержания когерентности. Один из них заключается в гарантировании получения вычислительным модулем исключительных прав доступа к строке памяти перед выполнением записи в эту строку. Такой тип протоколов называется протоколом записи **с аннулированием (write invalidate protocol)**, поскольку при выполнении записи в одну из копий другие копии аннулируются. Это наиболее часто используемый протокол как в схемах на основе справочников, так и в схемах наблюдения. Исключительное право доступа гарантирует, что во время выполнения записи не существует никаких других копий элемента данных, в которые можно писать или из которых можно читать, — все другие кэшированные копии элемента данных аннулированы.

Чтобы увидеть, как такой протокол обеспечивает когерентность, рассмотрим операцию записи, вслед за которой следует операция чтения другим ВМ. Поскольку запись требует исключительного права доступа, любая копия, поддерживаемая читающим ВМ, должна быть аннулирована (объявлена недействительной). При чтении аннулированной строки происходит промах кэш-памяти, который вынуждает выполнить выборку новой копии данных. Для выполнения операции записи можно потребовать, чтобы ВМ имел достоверную копию данных в своей кэш-памяти прежде, чем выполнять в нее запись. Таким образом, если оба ВМ попыта-

ются записать в одну и ту же строку памяти одновременно, один из них выиграет состязание у второго и вызовет аннулирование его копии. Другой ВМ для завершения своей операции записи должен сначала получить новую копию строки памяти, которая теперь уже должна содержать обновленное значение.

Альтернативой протоколу записи с аннулированием является обновление всех копий строки памяти в случае записи в нее. Этот тип протокола называется протоколом записи **с обновлением (write update protocol)** или протоколом записи **с трансляцией (write broadcast protocol)**. Обычно в этом протоколе для снижения трафика через коммуникационную среду при операции записи в каком-либо ВМ отслеживается, содержится ли строка памяти в кэшах других ВМ. Если нет, то трансляции обновленных данных не происходит.

Ниже приводятся два альтернативных случая, иллюстрирующих преимущества и недостатки каждого из двух рассмотренных подходов.

1. Несколько последовательных операций записи в одну и ту же строку памяти, не перемежающихся операциями чтения, требуют нескольких операций трансляции при использовании протокола записи с обновлением, но только одной начальной операции аннулирования при использовании протокола записи с аннулированием.

2. Задержка между записью строки памяти в одном ВМ и чтением записанного значения другим ВМ обычно меньше при использовании схемы записи с обновлением, поскольку записанные данные немедленно транслируются в читающий ВМ. Для сравнения, при использовании протокола записи с аннулированием в читающем ВМ сначала произойдет аннулирование его копии, затем будет производиться чтение данных и приостановка ВМ до тех пор, пока обновленная копия строки не будет доставлена в читающий ВМ.

Напомним, что существуют два способа реализации внесения изменений данных в основную память. Первый способ предполагает внесение изменений в основную память сразу после их возникновения в кэш-памяти или ОП другого ВМ. При этом вычислительный процесс в записывающем ВМ приостанавливается до завершения записи в основную память. Данный подход называется **сквозной записью**.

Другой способ предполагает отображение изменений в основной памяти либо в момент вытеснения строки из кэша записывающего ВМ, либо в момент возникновения запроса на чтение измененных данных в другом ВМ. Если копия строки памяти, в которую производится запись, находится в кэше (ОП) записывающего ВМ, то запись производится в кэш-память (ОП) этого ВМ. При отсутствии копии в кэш-памяти (ОП) записывающего ВМ производится запись в основную память. Указанный способ получил название **обратная запись**.

Обратная запись создает меньшую нагрузку на коммуникационную среду и обеспечивает большую производительность, однако контроллер кэш-памяти с

обратной записью значительно сложнее по сравнению с памятью со сквозной записью.

В качестве иллюстрации рассмотрим два распространенных протокола поддержки когерентности памяти, один из которых применяется для систем с сосредоточенной памятью, а другой — для систем с распределенной памятью.

5.7.3.3. Протокол MESI

Для обеспечения когерентности памяти для ВС с сосредоточенной общей памятью часто используется протокол **MESI**. MESI подразумевает использование кэш-памяти с обратной записью и является протоколом на основе наблюдения с аннулированием. Поскольку наблюдение предполагает использование в качестве коммутатора общей шины, то применение данного протокола ограниченно SMP-системами с шинной организацией коммуникационной среды.

Предполагается, что данные считываются из основной памяти в кэш некоторыми блоками фиксированного размера — строками. Строка, размещенная в кэш-памяти одного из ВМ, называется кэш-строкой, а сам ВМ по отношению к размещенным в нем кэш-строкам — локальным. Соответственно остальные ВМ будут называться удаленными.

Протокол MESI приписывает каждой кэш-строке одно из четырех состояний, начальные буквы названий которых и составляют аббревиатуру MESI. Строка может находиться в состоянии **Modified** («измененная»), **Exclusive** («монопольно копированная»), **Shared** («разделяемая») или **Invalid** («недействительная»). Состояние кэш-строки может быть изменено процессором локального ВМ посредством операций чтения, записи или внутренней проверки, а также удаленными ВМ или контроллером кэш-памяти посредством внешней проверки. Рассмотрим состояния MESI подробнее.

Измененная (M): данные в кэш-строке, помеченной как измененная, находятся только в кэш-памяти локального ВМ. Чтение и запись строки осуществляются локальным ВМ без доступа к общей шине.

Монопольно копированная (E): кэш-строка, как и M-строка, хранится только в кэш-памяти локального ВМ, однако данные в строке не изменялись локальным ВМ и идентичны данным, хранящимся в основной памяти. Монопольно копированная кэш-строка может быть считана или записана без выполнения цикла общей шины. После перезаписи кэш-строка отмечается как измененная.

Разделяемая (S): разделяемая кэш-строка может храниться в кэш-памяти нескольких ВМ. Запросы на чтение локальным ВМ выполняются через кэш-память. Запросы на запись к разделяемой кэш-строке всегда переключаются на общую шину, поэтому строки в кэш-памяти удаленных ВМ объявляются недействительными. Эта операция производится в цикле внешней проверки с использованием адреса, выдаваемого при выполнении цикла шины. При этом содержимое основной памяти также обновляется. Операция записи в локальную кэш-память обновляет только ее содержимое, которое недействительным не объявляется.

Недействительная (I): кэш-строка, отмеченная как недействительная, становится логически недоступной. Это может быть в том случае, если кэш-строка пуста или содержит недействительный элемент, т. е. не была обновлена. Все попытки обращения к недействительной кэш-строке интерпретируются как кэш-промах. В случае запроса на чтение контроллер кэш-памяти обычным образом инициирует заполнение кэш-строки. Запрос на запись переключается на общую шину и выполняется как сквозная запись.

В табл. 5.1 показано, каким образом размещаются данные в зависимости от состояния кэш-строки.

Таблица 5.1

Размещение данных в зависимости от состояния кэш-строки

Состояние MESI	*Значение кэш-строки*	*Значение в основной памяти*	*Есть копия в другом кэше?*	*Запрос на запись обращается к*
M	Действительно	Недействительно	Нет	Кэш-памяти
E	Действительно	Действительно	Нет	Кэш-памяти
S	Действительно	Действительно	Возможно	Кэш/ОП
I	Недействительно	Неизвестно	Возможно	ОП

Таблица 5.2 содержит схему перехода кэш-строки из одного состояния в другое. Состояние может поменяться в результате чтения или записи кэш-строки локальным ВМ, а также в ходе циклов проверки, инициируемых операциями чтения и записи кэш-строки удаленными ВМ. В цикле проверки может использоваться бит **INV** принудительного перевода кэш-строки в состояние **I**.

Таблица 5.2

Переходы кэш-строки из одного состояния в другое

Исходное состояние	*Состояние после чтения*	*Состояние после записи*	*Состояние после проверки*	
			INV = 0	INV = 1
I	Чтение строки из ОП, *E*	Сквозная запись в ОП, *I*	*I*	*I*
S	Чтение из кэша, *S*	Сквозная запись в ОП, *E*	Строка не обновлялась, *S*	Строка была обновлена удаленным ВМ, *I*
E	Чтение из кэша, *E*	Запись в кэш, *M*	Строка прочитана удаленным ВМ, *S*	Строка была обновлена удаленным ВМ, *I*
M	Чтение из кэша, *M*	Запись в кэш, *M*	Строка запрошена на чтение удаленным ВМ; обратная запись строки, *S*	Строка запрошена на запись удаленным ВМ; обратная запись строки, *I*

Рассмотрим диаграмму переходов кэш-строки из одного состояния в другое (рис. 5.27). Смена состояния *I* возможна только после чтения локальным ВМ строки из основной памяти в кэш. При этом состояние кэш-строки устанавливается в *S*, если копии прочитанной строки есть в кэш-памятях удаленных ВМ, и в *E*, если кэш-строка скопирована монопольно. Запись *I*-строки осуществляется непосредственно в основную память и не меняет ее состояния.

Когда кэш-строка находится в состоянии *S*, чтение ее как локальным, так и удаленными ВМ, не изменяют этого состояния. Локальный ВМ осуществляет чтение из кэша без обращения к основной памяти. Запись строки локальным ВМ приводит к сквозной записи в основную память, записи в локальную кэш-память и аннулированию (переводу в состояние *I*) копий строки во всех удаленных ВМ. Кэш-строка переводится в состояние *E*. Наоборот, запись *S*-строки удаленным ВМ приводит к ее аннулированию в локальном модуле и переходу в состояние *I*.

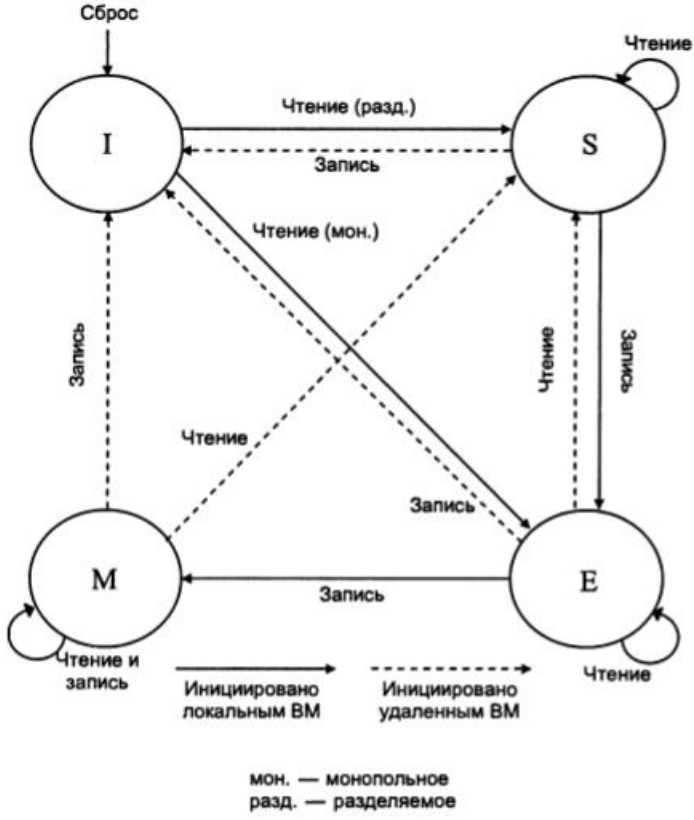


Рис. 5.27. Диаграмма переходов протокола MESI

Чтение монопольно-копированной строки (состояние *E*) локальным ВМ осуществляется из кэш-памяти и не меняет ее состояния. Чтение же копии этой

строки удаленным ВМ из основной памяти переводит кэш-строку в *S*-состояние. Запись *E*-строки удаленным ВМ делает ее недействительной (правильная копия строки будет находиться только в основной памяти). Запись *E*-строки локальным ВМ осуществляется только в кэш-память и переводит кэш-строку в состояние *M*.

Кэш-строка в состоянии *M* читается и записывается локальным ВМ без изменения состояния и обращения к основной памяти. Запрос на чтение этой строки со стороны удаленного ВМ приводит к обратной записи строки в основную память, чтению строки из основной памяти в кэш удаленного ВМ и переводу кэш-строки в состояние *S*. Запрос на запись со стороны удаленного ВМ также вызывает обратную запись измененной кэш-строки в основную память, после чего кэш-строка становится недействительной.

MESI — не единственный существующий протокол поддержки когерентности на основе наблюдения. Первым подобным протоколом является протокол однократной записи (write-once), предложенный в 1983 г. Гудменом. Кроме этого, в разное время были разработаны и реализованы в разных ВС протоколы Synapse, Berkley, Illinois, Firefly и др.

5.7.3.4. Протоколы поддержки когерентности для распределенной общей памяти

Рассмотрим вопросы поддержания когерентности памяти в NUMA-системе. Очевидно, при каждом промахе в кэш в любом ВМ инициируется запрос требуемой строки из того ВМ, в памяти которого эта строка размещена. В дальнейшем ВМ, содержащий некоторую строку памяти, по отношению к этой строке будем называть **резидентным**, а строку памяти по отношению к содержащему ее ВМ — **резидентной**. ВМ, не содержащие некоторую строку памяти, по отношению к этой строке будем называть **удаленными**, а саму строку по отношению к этим ВМ — **удаленной**.

Для поддержания когерентности памяти в NUMA-системах часто применяется протокол DASH, который рассмотрим подробнее.

Протокол **DASH (Directory Architecture for SHared memory** — «архитектура для общей памяти на основе справочника») был создан в Стенфордском университете в 1992 г. Как следует из названия, протокол DASH в отличие от MESI основывается на справочнике, а не на наблюдении.

Каждый ВМ для каждой резидентной строки ведет список модулей, в кэшах которых размещена ее копия.

С каждой строкой в резидентном для нее модуле связаны три ее возможных **глобальных** состояния:

1) «**некэшированная**», если копия строки не находится в кэше какого-либо удаленного ВМ;

2) «**удаленно-разделенная**», если копии строки размещены в кэшах удаленных ВМ;

3) «**удаленно-измененная**», если строка изменена операцией записи в каком-либо удаленном ВМ.

Кроме этого, каждая кэш-строка в удаленном ВМ находится в одном из трех **локальных** состояний:

1) «**невозможная к использованию**», если копии строки нет в кэше, или копия аннулирована;

2) «**разделяемая**», если в кэше есть неизмененная копия, которая, возможно, размещается также в кэшах других ВМ;

3) «**измененная**», если копия изменена операцией записи.

Каждый ВМ может читать из своего кэша, если состояние читаемой кэш-строки «разделяемая» или «измененная». Если строка находится в состоянии «невозможная к использованию», то в резидентный ВМ посылается запрос «промах чтения».

Если глобальное состояние строки в резидентном модуле «**некэшированная**» или «**удаленно-разделенная**», то копия строки посылается в запросивший удаленный модуль; сам запросивший модуль вносится в список ВМ, содержащих копии рассматриваемой строки.

Если глобальное состояние строки «**удаленно-измененная**», то запрос «**промах чтения**» перенаправляется в удаленный ВМ, содержащий измененную строку. Этот модуль пересылает требуемую строку в запросивший модуль и в резидентный модуль. В резидентном модуле для этой строки устанавливается глобальное состояние «**удаленно-разделенная**».

Если удаленный ВМ выполняет операцию записи в кэш-строку с локальным состоянием «**измененная**», то запись выполняется, и вычисления продолжаются. Если локальное состояние записываемой кэш-строки «**невозможная к использованию**» или «**разделяемая**», то в резидентный для строки модуль посылается запрос на захват строки в монопольное использование. Выполнение записи приостанавливается до получения подтверждений, что все удаленные ВМ, разделяющие рассматриваемую строку, перевели ее копии в состояние «**невозможная к использованию**».

Если глобальное состояние строки в резидентном модуле «**некэшированная**», то строка отсылается запросившему удаленному ВМ, и этот модуль продолжает приостановленные вычисления.

Если глобальное состояние строки «**удаленно-разделенная**», то резидентный модуль рассылает по списку всем ВМ, имеющим копию строки, запрос на перевод этих строк в состояние «**невозможная к использованию**». По получении этого запроса каждый из удаленных ВМ изменяет локальное состояние своей копии строки на «**невозможная к использованию**» и посылает подтверждение исполнения в модуль, инициировавший операцию записи. В записывающем ВМ строка после выполнения записи переходит в состояние «**измененная**». При этом

в резидентном модуле глобальное состояние строки устанавливается в состояние **«удаленно-измененная»**.

Заметим, что возможна реализация протокола DASH и в SMP-системах с шинной организацией. Перенос же протоколов на основе наблюдения в системы с распределенной общей памятью невозможен по причине отсутствия широковещательной магистрали данных.

Среди прочих протоколов на основе справочника можно назвать Tang, Censier, Archibald и др.

5.7.4. Массово-параллельные ВС

5.7.4.1. Общие принципы построения MPP-систем

Альтернативой ВС с общей памятью являются **системы с массовым параллелизмом (Massively Parallel Processor — MPP)**, основанные на концепции параллельных вычислительных модулей. Вычислительная система здесь представляет собой совокупность вычислительных модулей, связанных друг с другом с помощью высокоскоростных линий связи. Каждый ВМ имеет собственную оперативную память и управляется собственной операционной системой.

Процессор, его оперативная память, интерфейс межпроцессорных связей в совокупности и составляют вычислительный модуль. Высокоскоростная линия связи, по принципу «точка-точка» связывающая два ВМ между собой, называется **линком**.

Как правило, объединенные вычислительные модули MPP-системы образуют распределенный составной коммутатор с тем или иным графом межмодульных связей. Совокупность объединенных ВМ часто называют **решающим полем**, или **вычислителем**. Ввод-вывод информации и другое общение вычислителя с внешней средой осуществляется через одну или несколько специально выделенных **управляющих ЭВМ** (рис. 5.28).

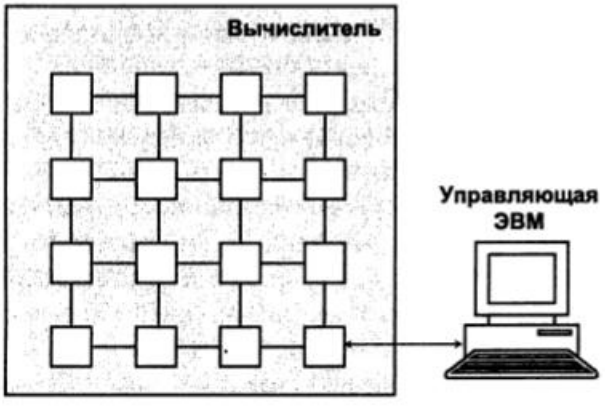


Рис. 5.28. Обобщенная структура MPP-системы

Кроме обеспечения ввода-вывода, управляющая ЭВМ в соответствии со своим названием осуществляет функции управления, в том числе начальную загрузку вычислителя, выделение и освобождение вычислительных модулей для нужд параллельных задач, ведение очереди задач и т. п.

К явным преимуществам массово-параллельных ВС относятся масштабируемость и гибкость топологии — ВС, построенные по MPP-технологии, содержат сотни и даже тысячи ВМ, которые могут объединяться по самым различным топологическим схемам. Теоретически число ВМ в MPP-системе не ограничено, что позволяет разработчикам подобных систем добиваться самых высоких показателей пиковой производительности.

Тем не менее по сравнению с ВС с общей памятью MPP-системы обладают двумя существенными недостатками:

1) коммуникационная среда MPP-систем обладает значительно худшими характеристиками пропускной способности и латентности;

2) возникают значительные сложности при программировании задач, которым необходима память, превышающая размер локальной оперативной памяти одного ВМ.

Указанные недостатки приводят к тому, что для массово-параллельных ВС фактически доступны только модели программирования «процесс/канал» или «обмен сообщениями», что существенно суживает класс задач, которые могут быть успешно решены на данных системах.

Принято считать, что промышленное освоение MPP-систем началось в конце 80-х годов XX в. с выпуска британской фирмой Inmos **транспьютеров T-2xx**, **T-4xx**, **T-8xx**. Транспьютер представлял собой 32-разрядный RISC-процессор, имевший в своем составе 4 высокоскоростных дуплексных линка, с помощью которых было возможно объединение друг с другом большого числа транспьютеров.

Революционной особенностью транспьютера являлась аппаратная поддержка модели программирования «процесс/канал». При этом для программиста не делалось разницы, где находятся обменивающиеся информацией процессы – на одном или разных транспьютерах, что позволяло разрабатывать легко масштабируемые программы. Добавим, что в транспьютерных системах широко применялся уже упоминавшийся (см. п. 5.4.4.1) язык высокого уровня OCCAM.

С середины 90-х годов XX в. и по сегодняшний день MPP-системы активно развиваются, занимая высшие места в списке 500 наиболее производительных компьютеров мира.

Примерами реализации MPP-архитектуры могут служить суперкомпьютеры **Cray-T3E** компании Cray Research, **SP2** фирмы IBM, а также **ASCI Red** фирмы Intel, первым перешагнувший в 1997 г. терафлопный рубеж производительности. В модели ASCI Red в качестве вычислительных модулей использовались стандартные аппаратные средства — двухпроцессорные материнские платы Pentium

Pro, снабженные специальными коммуникационными интерфейсами для связи с другими модулями. Пиковая производительность образца из 9152 процессоров составляла 1,8 TFLOPS, а его стоимость — 46 миллионов долларов. Данная супер-ЭВМ была предназначена для моделирования эффекта старения ядерных боеголовок американских стратегических ракет.

Среди последних разработок следует отметить системы **Blue Gene** компании IBM (более 30 тыс. процессоров, производительность 180 TFLOPS) и **Cray XT** фирмы Cray Research.

5.7.4.2. Отечественные массово-параллельные ВС МВС-100х

Примером отечественных MPP-систем могут служить многопроцессорные вычислительные системы МВС-100 и МВС-1000 (МВС-100х). Обе системы имеют схожую архитектуру и состоят из двух главных частей — **вычислителя** и **управляющей ЭВМ** (рис. 5.29).

Вычислитель представляет собой совокупность двухпроцессорных вычислительных модулей. Один из процессоров модуля выполняет вычислительные функции и называется **вычислительным процессором (ВП)**. На другой процессор возложены функции информационного обмена, поэтому он называется **связным (СП)**. Процессоры ВМ имеют как общую разделяемую, так и свою собственную оперативную память.

Каждый СП имеет несколько линков. Посредством линков вычислительные модули объединяются в сеть. Все модули вычислителя имеют в системе уникальный **физический номер** от **0** до *N*. Один из линков нулевого модуля соединен со специальным адаптером (**адаптером линка**), вставленным в управляющую ЭВМ, чем обеспечивается связь последней с вычислителем. Нулевой модуль вычислителя называется поэтому **корневым**.

Рассмотрим состав и характеристики многопроцессорных вычислительных систем МВС-100 и МВС-1000.

Многопроцессорная вычислительная система **МВС-100** была разработана НИИ «Квант» и рядом институтов РАН в 1994 г. Проектом предусматривалось достижение в максимальной конфигурации разрабатываемой вычислительной системы производительности до 100 млрд операций с плавающей точкой в секунду (100 GFLOPS).

За почти десятилетие эксплуатации получен обширный практический опыт использования супер-ЭВМ МВС-100 для решения сложных научно-прикладных задач. На испытаниях и в реальной эксплуатации система МВС-100 показала высокую эффективность и хорошую надежность при относительно невысокой стоимости.

В качестве вычислительного процессора во всех моделях семейства МВС-100 использовался 64-разрядный микропроцессор фирмы Intel A80860XR (i860) 40 Мгц, а в качестве связного — транспьютер фирмы Inmos T805 25 Мгц. Каждый вычислительный модуль содержал до 16 Мбайт разделяемой памяти, доступной

как транспьютеру, так и i860, и до 4 Мбайт локальной памяти транспьютера, доступной только транспьютеру. Каждый транспьютер имел по четыре линка скоростью до 20 Мбит/с.

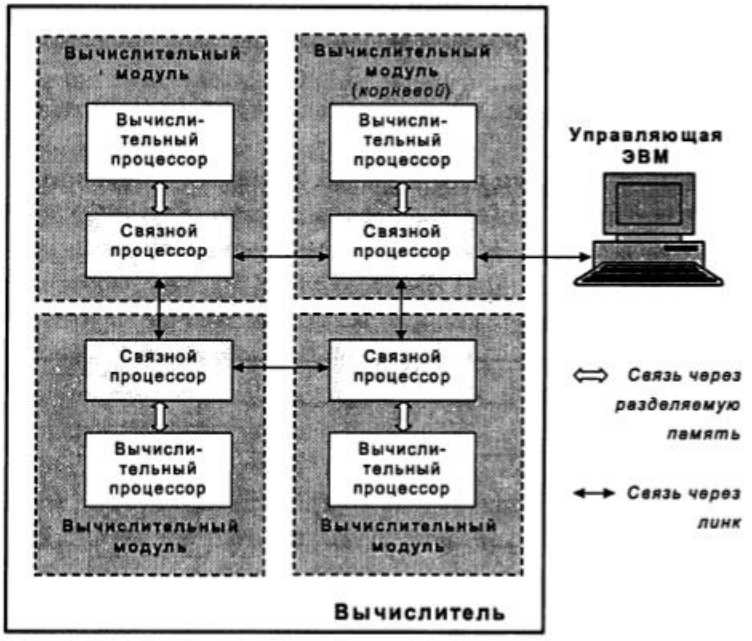


Рис. 5.29. Архитектура многопроцессорных систем МВС-100 и МВС-1000

Объединенные друг с другом вычислительные модули представляли собой решающее поле. Кроме решающего поля, в состав супер-ЭВМ МВС-100 могли входить до 8 управляющих рабочих станций, объединенных в локальную сеть, платы сопряжения управляющих рабочих станций с решающим полем, а также рабочие станции, подключавшиеся либо непосредственно к одному из ВМ решающего поля, либо к локальной сети управляющих ЭВМ. Эти последние рабочие станции использовались для подготовки исходных данных для счета на МВС-100 и для представления результатов.

Вычислитель МВС-100 делился на секции по несколько ВМ, каждая из которых имела собственную линию аппаратного сброса.

Как уже было сказано, вычислительные модули соединялись друг с другом посредством транспьютерных линков. При этом в основу построения принятой архитектуры МВС-100 была положена оригинальная модульная структура, позволяющая совместить достоинства двух известных топологических схем — «решетки» и «гиперкуба» (рис. 5.30).

Несмотря на то что на вычислительном процессоре отсутствовала операционная система, МВС-100 обладала широким набором инструментальных программных средств. Как основное средство информационного обмена следует отметить

операционную среду Router, обеспечивавшую информационные обмены через транспьютерную сеть. Операционная среда Router включала в себя одноименную библиотеку передачи сообщений. Данная библиотека относилась к классу низкоуровневых библиотек и предоставляла пользователю набор примитивов для организации информационного обмена между двумя любыми процессорами системы по модели Хоара.

В качестве операционной системы управляющей ЭВМ использовалась Unix-подобная ОС Helios.

Среди средств разработки параллельных программ, имевшихся на вооружении МВС-100, находились упоминавшиеся языковые средства GNS, Fortran DVM, непроцедурный язык Норма. Имелась система коллективного и удаленного доступа к вычислителю МВС-100. В ИММ УрО РАН была разработана и внедрена в эксплуатацию система планирования прохождения заданий на МВС-100.

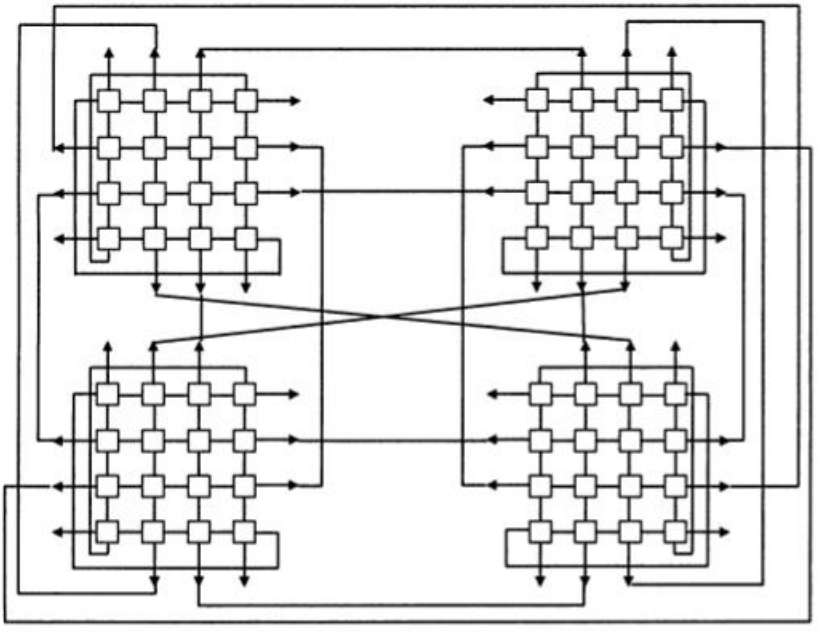


Рис. 5.30. Топология МВС-100 для 64-процессорного образца

Идеи, положенные в основу построения МВС-100, в своем развитии нашли воплощение в разработке высокопроизводительной вычислительной системы **МВС-1000** (1998).

Архитектура МВС-1000 подобна архитектуре МВС-100 в смысле построения супер-ЭВМ как MPP-системы, состоящей из совокупности связанных между собою двухпроцессорных вычислительных модулей. Структура ВМ МВС-1000 аналогична структуре ВМ МВС-100, только в качестве вычислителя вместо i860 использовался серийный микропроцессор Alpha-21164 300-500 МГц фирмы DEC

(позже — Compaq, HP). В качестве связного процессора вместо транспьютеров использовались сигнальные процессоры — либо TMS320C44 фирмы Texas Instruments, либо ADSP21060 (Sharc) фирмы Analog Devices. Оба сигнальных процессора имели по несколько (С44 – четыре, Sharc — шесть) высокоскоростных линков. Это позволяло комплексировать вычислительные модули и наращивать производительность системы способом, аналогичным применявшемуся в МВС-100.

Программное обеспечение МВС-1000 сделало значительный шаг вперед по сравнению с МВС-100. Во-первых, на вычислительном процессоре появилась POSIX-совместимая ОС VxWorks, что значительно облегчило разработку прикладных программ. Так же как и в МВС-100, на сети связных процессоров функционировала операционная среда Router, дополненная функциями защиты, обеспечения ввода-вывода и управления вычислительными ресурсами. Кроме этого, МВС-1000 впервые в истории отечественных параллельных ВС была обеспечена коммуникационной библиотекой, полностью реализовывавшей стандарт MPI.

Одной из важнейших вновь разработанных составляющих системного ПО МВС-1000 явилась разработанная в ИПМ им. М.В. Келдыша РАН Система управления прохождением задач (СУПЗ), обеспечившая многопользовательский и многозадачный режим работы системы с применением безопасного удаленного доступа.

5.7.5. Кластерные ВС

Кластерная ВС, как и MPP-система, представляет собой совокупность вычислительных модулей (возможно, многопроцессорных), объединенных некоторой коммуникационной средой. Как и в MPP, каждый ВМ кластерной ВС имеет собственную оперативную память и управляется своей ОС. Как и MPP, кластерные ВС (или просто кластеры) могут наращиваться до нескольких тысяч процессоров. *Отличие массово-параллельных ВС от кластеров состоит в том, что построение последних осуществляется исключительно на базе серийных промышленных комплектующих, в то время как те или иные компоненты MPP представляют собой оригинальные разработки.*

В качестве вычислительных модулей в кластерах используются готовые ЭВМ или SMP-система малого размера. В качестве коммуникационной среды — стандартное сетевое оборудование массового и среднесерийного выпуска. За счет применения стандартных промышленных комплектующих кластерные ВС имеют наивысший показатель «производительность/стоимость» среди всех параллельных архитектур.

Предпосылкой для появления во второй половине 90-х годов XX в. кластерной архитектуры явилась так называемая «**вторая суперкомпьютерная революция**», связанная в первую очередь с двумя достижениями в области аппаратуры крупносерийного выпуска. Повсеместное оснащение персональных компьютеров высоко-

скоростной шиной PCI, с одной стороны, и переход на дешевую, но довольно быструю, причем коммутируемую, сеть Fast Ethernet — с другой, привели к тому, что кластеры по своим коммуникационным возможностям вплотную приблизились к специализированным MPP. При этом среди производителей специализированного коммуникационного оборудования, такого, которое раньше применялось только в составе MPP, стало принято оформлять свои изделия в виде стандартных плат PCI, устанавливаемых в компьютеры общего назначения, а не в виде наборов микросхем, из которых требовалось изготавливать вычислительные модули.

В этих условиях изготовление MPP-систем из специализированных, только для этой цели спроектированных и изготовленных, модулей во многом потеряло смысл. За счет некоторых потерь в плотности компоновки стало возможно изготовить многопроцессорную ВС не из специальных модулей, а из стандартных персональных компьютеров или рабочих станций, и при необходимости оснастить кластер высокопроизводительной, специализированной коммуникационной средой. Разница между MPP и кластером, таким образом, из области аппаратной постепенно сместилась в область организационную.

Технология создания кластерных ВС развивается в рамках концептуально близких направлений: *Network of Workstations* (NOW, «сеть рабочих станций») и *Pile-of-PCs* (PopC, «набор ПК»).

Технология **NOW** ведет начало от разработок Калифорнийского университета Berkeley по созданию вычислительного кластера с одноименным названием. Целью данного проекта являлось объединение ресурсов вычислительной сети, состоящей из рабочих станций, в систему, подобную супер-ЭВМ, с централизованным управлением и быстрыми связями между вычислительными узлами.

При создании кластера использовались стандартные программные и аппаратные средства (стандартные рабочие станции в качестве вычислительных узлов кластера, стандартная операционная система Solaris в вычислительных узлах, сетевые ATM-платы и коммутаторы в локальной сети кластера). Аппаратная конфигурация кластера NOW включала 105 рабочих станций SUN Ultra 170, соединенных сетью Myrinet.

В настоящее время отличительными особенностями технологии NOW являются:

- использование в качестве ВМ высокопроизводительных, возможно многопроцессорных, рабочих станций;
- исполнение всех ВМ в виде так называемых стандартных по размерам blade-модулей («модулей-лезвий»), вставляемых в стандартные стойки («шасси»);
- применение в качестве коммуникационной среды высокопроизводительного сетевого оборудования средней серийности (Myrinet, SCI, InfiniBand и др.);
- возможно использование дорогого лицензированного системного ПО (ОС, трансляторов, коммуникационных библиотек, отладчиков и т. п.).

С появлением на рынке персональных компьютеров с высокопроизводительными микропроцессорами, сравнимых по производительности с рабочими станциями, получило развитие второе направление создания кластерных систем — на базе персональных компьютеров — **PopC**.

Отличиями концепции PopC являются следующие:

- использование в качестве ВМ массово выпускаемых персональных компьютеров;
- как правило, отсутствие специализированного «стоечного» исполнения, ВМ используются в стандартных корпусах ПК;
- в качестве коммуникационной среды используется массово выпускаемое сетевое оборудование (FastEthernet, GigaEthernet);
- использование исключительно свободно распространяемого ПО.

Первым проектом, в рамках которого были реализованы основные принципы PopC, являлся **Beowulf**. Этот проект был открыт в 1994 г. под патронажем NASA в интересах географических и космических исследований, проводимых центром GSFC (Goddard Space Flight Center). Работы над проектом Beowulf проводились научным центром CESDIS (Center of Excellence in Space Data and Information Sciences) совместно с компанией Red Hat Software.

Первый кластер (16 компьютеров на базе Intel 80486) в рамках программы Beowulf был создан в 1994 г. и получил название *Wiglaf*. Однако *всемирную известность* проект приобрел в 1998 г. после громкого успеха 68-компьютерного кластера **Avalon**, созданного в национальной лаборатории Los Alamos (США). Каждый узел кластера представлял собой рабочую станцию DEC Alpha с процессором Alpha 21164A 533 МГц, оперативной памятью 128 Мбайт, жестким диском 3 Гбайт и адаптером локальной сети Fast Ethernet. В кластере были использованы четыре 36-портовых Fast Ethernet коммутатора, к каждому из которых было добавлено по два модуля Gigabit Ethernet, подключенных к 12-портовому Gigabit Ethernet коммутатору.

Стоимость кластера Avalon составила 152 тысячи долларов. На тестах Linpack система показала производительность 19,7 GFLOPS, что на тот день стало сравнимо с производительностью 64-процессорного SMP-суперкомпьютера SGI Origin 2000 (20.1 GFLOPS), цена которого составляла около двух миллионов долларов. Интересно, что полная сборка и настройка кластера из отдельных рабочих станций была осуществлена всего за 3 дня.

Среди отечественных разработок следует выделить систему МВС-1000М, созданную в НИИ «Квант» по заказу Межведомственного суперкомпьютерного центра РАН в 2001 г. МВС-1000М — первая отечественная система, перешагнувшая терафлопный рубеж пиковой производительности.

Отметим, что хотя кластерные ВС при сравнительно небольшой стоимости и способны достигать высоких показателей пиковой производительности, характеристики стандартного сетевого оборудования и шины PCI все-таки хуже, чем

у специализированных MPP-систем и тем более, чем у NUMA- и SMP-систем. Таким образом, все недостатки MPP-систем, только в несколько большей степени, распространяются и на кластерные ВС.

5.8. УПРАВЛЕНИЕ ВЫЧИСЛИТЕЛЬНЫМИ РЕСУРСАМИ В ПАРАЛЛЕЛЬНЫХ ВС

Нетрудно заметить, что параллельные ВС имеют значительно более сложную организацию, нежели ЭВМ последовательного типа. Задача управления вычислительными ресурсами, в ЭВМ последовательного типа целиком возлагаемая на операционную систему, в параллельных ВС, в силу их сложности, обычно распределяется по разным программным и аппаратным компонентам. Рассмотрим условия, в которых эксплуатируются современные параллельные ВС, и требования к подсистеме управления вычислительными ресурсами, вытекающие из этих условий.

Прежде всего следует учитывать, что параллельные ВС имеют значительно меньшую серийность, чем обычные ЭВМ и ВС, и, как следствие, являются достаточно дорогостоящими вычислительными установками. Столь дорогие по цене ресурсы не должны использоваться монопольно, из чего вытекает требование обеспечения **коллективного (многопользовательского) доступа** к параллельной ВС. Кроме этого, учитывая распространенность глобальной вычислительной сети Интернет, пользователям параллельной ВС должна быть предоставлена возможность **удаленного доступа** к ВС. В свою очередь, многопользовательский режим работы с предоставлением удаленного доступа требует обеспечения всесторонней **защиты от несанкционированного доступа (НСД)**.

Как правило, современные пользователи параллельных ВС работают с несколькими параллельными задачами (вариантами одной задачи) одновременно, при этом далеко не каждая из них может требовать для своего исполнения всего решающего поля целиком. Во избежание простоя вычислительных ресурсов управляющая подсистема должна обеспечивать **мультизадачный режим** функционирования ВС, подразумевающий *деление решающего поля между разными задачами разных пользователей*.

Каждая параллельная программа требует для своего исполнения некоторый **параллельный вычислительный ресурс**, под которым будем понимать *подсистему связанных между собой процессоров решающего поля, выделяемый задаче на определенное время*. Величина требуемого параллельного ресурса (число процессоров и время исполнения) определяет **размер параллельной задачи**. Множество различных параллельных задач от разных пользователей образуют **входной поток заданий**. Велика вероятность возникновения ситуации, когда для очередной задачи из входного потока в определенный момент времени требуемый параллельный ресурс будет занят. В этом случае управляющая система должна обеспечить **ведение очереди параллельных задач**.

После прохождения через очередь требуемый параллельный ресурс должен быть **выделен** задаче. Как правило, при постановке задачи в очередь от пользователя требуется специфицировать время ее выполнения. При завершении задачи или при исчерпании специфицированного времени выделенный параллельный ресурс должен быть **освобожден** от процессов задачи.

Во время исполнения параллельной задачи управляющая система должна предоставить ей возможность осуществления **операций ввода-вывода**, в первую очередь **файлового**. С учетом сложной топологической структуры современных параллельных ВС данная задача является не столь тривиальной, как это может показаться на первый взгляд. Например, для ВС, построенных на основе распределенного составного коммутатора, трафик, порождаемый операциями ввода-вывода, должен транслироваться к управляющей ЭВМ через несколько промежуточных вычислительных модулей. При этом управляющая подсистема должна совмещать трафик ввода-вывода с передачей сообщений, которыми обмениваются ветви параллельной задачи.

В многопроцессорных системах на основе общей памяти организация ввода-вывода также имеет свои сложности, связанные с тем, что необходимо распределять выполнение операций ввода-вывода и обработку прерываний по нескольким процессорам системы.

В конечном итоге главной задачей параллельной ВС при выполнении параллельной программы является достижение высокой реальной производительности, близкой к пиковой. Значительный вклад в итоговый показатель производительности вносит управляющая система, от которой требуется **эффективная организация информационного обмена** между различными ветвями параллельной задачи. Кроме этого, нетрудно видеть, что реальная производительность зависит от того, насколько равномерно распределена вычислительная нагрузка по процессорам параллельной ВС. **Обеспечение равномерной загрузки процессоров вычислителя** — так называемая **балансировка загрузки** — также является одним из важных требований к современной управляющей системе параллельных ВС. Эффективные методы решения этой задачи в настоящее время разработаны только для ВС с общей памятью, для остальных ВС, особенно гетерогенных, балансировка загрузки представляет собой трудно решаемую проблему. Связано это с тем, что эффективное перемещение контекста программы с одного процессора на другой невозможно в слабосвязанных ВС.

Другим часто выдвигаемым требованием, которому должна удовлетворять управляющая система, является **решение представленного набора параллельных задач к заданному времени**. Эффективность выполнения данного требования напрямую зависит от качества планирования очереди.

Наконец, современную параллельную ВС невозможно представить без организации **отказоустойчивых вычислений**. Отказоустойчивость достигается, во-первых, за счет постоянного наблюдения за техническим состоянием вычис-

лительных ресурсов (осуществления так называемого **мониторинга** вычислительных ресурсов), во-вторых, за счет применения эффективных средств **восстановления после отказов**.

Таким образом, можно сформулировать **задачи современной управляющей системы** параллельных ВС.

1. Обеспечение безопасного удаленного доступа многих пользователей к ВС.
2. Прием входного потока параллельных задач разной сложности и размеров от разных пользователей.
3. Ведение очереди параллельных задач с учетом требований полной загрузки вычислителя и решения заданного набора задач к заданному времени.
4. Выделение необходимых вычислительных ресурсов для нужд параллельной задачи.
5. Освобождение выделенных вычислительных ресурсов после окончания счета параллельной задачи.
6. Организация ввода-вывода и информационного обмена для многих ветвей выполняющейся параллельной задачи.
7. Обеспечение равномерной загрузки процессоров параллельной ВС.
8. Обеспечение мониторинга вычислительных ресурсов и восстановления после отказов.

Выполнение указанных задач возлагается на различные блоки управляющей системы, которые, вообще говоря, могут быть распределены по различным программным и аппаратным компонентам ВС. Заметим, что управляющие блоки могут быть выстроены в иерархическую многоуровневую систему.

Предполагается, что в параллельной ВС могут быть выделены отдельные подсистемы, и на каждую выделенную подсистему может быть назначен определенный вид вычислительных работ. Состояние параллельной ВС, связанное с определенным разбиением на подсистемы и назначением на подсистемы определенных видов работ, называется **функциональным состоянием** системы. Периодически в такой системе необходимо производить перераспределение ресурсов, т. е. производить изменение функционального состояния системы.

На вход системы поступает поток заданий на выполнение, которые могут ждать освобождения вычислительных ресурсов в одной или нескольких входных очередях. Вычислительные процессы, протекающие на выделенных подсистемах, могут находиться в разных состояниях (выполнения, ожидания, зацикливания) и выдавать разные запросы к ОС параллельной ВС. Текущее функциональное состояние системы, состояние вычислений процессов на выделенных подсистемах, текущее содержимое входных очередей определяют текущую **мультипрограммную ситуацию**.

Каждый из иерархических уровней управляющей системы уменьшает неопределенность сложной мультипрограммной ситуации, определяя и фиксируя

ряд параметров для вышестоящего уровня. Выделяются четыре уровня принятия решений.

Первый уровень — *уровень пользователя*, который выбирает алгоритмы для своих задач, средства для записи алгоритмов, их анализа, отладки и т. п.

На **втором уровне** принимаются локальные, автономные решения по распределению ресурсов в рамках *одного* вычислительного модуля.

На **третьем уровне** принимаются решения, которые связаны с распределением ресурсов подсистемы и доступа к системным (обобществленным) внешним устройствам. С появлением задач, которые не могут быть решены в рамках существующих подсистем и установленных режимов, вырабатываются требования к вышестоящему уровню для изменения функционального состояния параллельной ВС.

На **четвертом уровне** принимаются глобальные, системные решения, изменяющие функциональное состояние всей ВС. Эти решения определяются приоритетами режимов функционирования, очередями задач, состоянием ресурсов системы, директивами оператора и т. п.

Существуют три основных типа организации вычислительного процесса в параллельных ВС:

- ведущий — ведомый;
- раздельное выполнение заданий в каждом процессоре;
- симметричная, или однородная, обработка во всех процессорах.

Возможны также малоразличающиеся варианты такой организации процессов.

Наиболее простым для реализации является метод обработки типа «**ведущий-ведомый**». Он чаще всего используется в асимметричных и неоднородных ВС, таких, как ВС с распределенной ОП (массово-параллельные, кластерные и т. п.), но может применяться и в системах с общей памятью. При данной организации системные управляющие функции реализуются только в одном процессоре (ведущем), который может быть выбран оператором или автоматически. Этот процессор может иметь специально разработанную для управления конфигурацию или быть обычным универсальным процессором. Специально разработанный супервизорный процессор предназначается исключительно для выполнения управляющих функций, в то время как универсальный процессор может быть использован для расчетов по прикладным программам, когда супервизорные процедуры не выполняются.

Система типа «ведущий-ведомый» подвержена отказам при сбоях в ведущем процессоре. Если последний не является специализированным, то оператор может заново запустить систему, используя при этом в качестве ведущего другой процессор. Ведущий процессор должен обеспечить выполнение управляющих функций достаточно быстро, чтобы удовлетворять выдвигаемым требованиям с опережением. Если это условие не выполняется, то эффективность использования системы может оказаться очень низкой.

Заметим, что в массово-параллельных и кластерных ВС метод «ведущий-ведомый» фактически является единственно возможным методом организации

вычислительного процесса. В качестве ведущего процессора здесь выступает управляющая ЭВМ, на которую возлагаются все управляющие функции третьего и четвертого уровней иерархии. Функции первого и второго уровней выполняются ОС вычислительных модулей.

Другим типом организации обработки информации в параллельной ВС является **раздельное выполнение заданий** в каждом процессоре. Заметим, что применение данного типа обработки возможно уже только в ВС с общей памятью. В этом случае управляющие (супервизорные) функции осуществляются каждым процессором в соответствии с его потребностями и требованиями предназначенных для него программ. Поскольку модули супервизора выполняются несколькими процессорами, предусматривается их повторная входимость, или копии их должны быть загружены в каждый процессор. В некоторых случаях используется сочетание указанных двух методов. Число конфликтов, вызванных блокировкой системных таблиц, в таких системах невелико, так как каждый процессор может иметь собственный набор необходимых программных модулей. При этом число общих исполнительных таблиц незначительно.

Все операции ввода-вывода для любой задачи выполняет процессор, на который эта задача возложена. Прерывания по вводу-выводу поступают на процессор, который инициирует соответствующие операции ввода-вывода.

При отказе какого-либо процессора система с раздельным выполнением заданий в целом не теряет работоспособности. Однако восстановление и повторный запуск программ, которые выполнялись отказавшим процессором, обычно сопряжены с большими трудностями. Эффективность такой системы мала, если один процессор загружен большими заданиями, в то время как другие простаивают.

Третий тип организации обработки — **симметричная обработка** во всех процессорах, — так же, как и второй, используется только в ВС с общей памятью.

Симметричная обработка наиболее естественна для SMP-систем, поскольку такая ВС чаще всего является совокупностью одинаковых процессорных блоков, которые совместно используют общую память, а также каналы и устройства ввода-вывода.

Общесистемное управление в SMP-системе непрерывно перераспределяется между процессорами. Один из процессоров, обеспечивающий в данный момент управление системными таблицами и такие функции, как планирование, называется исполнительным. Для предотвращения конфликтных ситуаций в каждый момент времени исполнительным может быть только один процессор. Каждый процессор выполняет также управляющие функции, неразрывно связанные с решаемой им задачей, и функции, необходимые для выполнения новой задачи, если текущая будет прервана или полностью завершена.

Поскольку процессоры однородны и могут быть использованы по одному назначению, любая задача может обрабатываться одним или несколькими процессорами. Для успешного ее выполнения могут быть применены разные наборы процессоров. Лучшая балансировка загрузки и повышение эффективности системы

достигаются путем выполнения неспецифических функций, таких, как прерывание для ввода-вывода наименее загруженных в данный момент процессоров.

Несмотря на то что в общесистемном управлении только один процессор является исполнительным, несколько других могут одновременно выполнять те же управляющие процедуры. Поэтому эти процедуры должны обладать повторной входимостью или должны иметься копии для каждого случая запуска.

Подобная организация обладает несколькими важными преимуществами: при выходе из строя отдельных устройств система частично сохраняет работоспособность (т. е. происходит постепенная деградация); благодаря однородности возможно эффективное резервирование для обеспечения надежности; наиболее эффективно используются ресурсы.

Несмотря на наиболее совершенную форму организации функционирования параллельной ВС симметричная обработка в то же время наиболее сложна для реализации. Одна из главных проблем симметричной обработки тесно связана с необходимостью равномерной загрузки всех процессоров в каждый период времени. Решение этой проблемы для неделимых задач может потребовать применения сложных методов прогнозирования с целью оценки времени решения задач и других требований к ресурсам системы. В настоящее время не существует оптимального метода решения этой проблемы. Поэтому в современных ОС для SMP-систем широко применяется **мультитредовый (многопотоковый)** подход, в соответствии с которым ветви параллельной программы представляются в виде так называемых **нитей** (**тредов**, от английского **thread** — «нить») одного вычислительного процесса. Нити представляют собой отдельные потоки вычислений, выполняющиеся в адресном пространстве единого вычислительного процесса. За счет применения мультитредового подхода возможно достижение очень высоких показателей равномерности загрузки процессоров и масштабируемости всей параллельной ВС в целом.

5.9. ПАРАЛЛЕЛЬНЫЕ ВС С НЕТРАДИЦИОННОЙ АРХИТЕКТУРОЙ

5.9.1. Вычислительные системы, управляемые потоком данных

Рассмотренные выше параллельные ЭВМ и ВС имели так называемую традиционную архитектуру, которая опирается на два основополагающих взаимосвязанных принципа.

1. Процессоры параллельной ВС имеют фон-неймановскую архитектуру, предусматривающую хранение программного кода и данных в одной памяти и последовательное выполнение команд. Адрес следующей команды, которую процессор должен выполнить, определяется значением **программного счетчика**. Фактически порядок выполнения команд задается последовательностью значений программного счетчика. *Команда выполняется после того, как выполнена предшествующая ей команда последовательности.* При этом процессор считает каждую очередную машинную команду полностью готовой к исполнению и не-

зависимой от других команд выполняемой программы. Задача обеспечения действительной готовности команды, т. е. правильности кода операции и операндов, целиком возлагается на программиста.

2. Построение **схемы параллелизма** (распределение вычислений и данных по процессорам и организация обмена данными) в традиционных параллельных ВС также целиком возлагается на программиста. Как уже отмечалось, та или иная парадигма или модель параллельного программирования представляет собой не что иное, как способ записи уже готовой схемы параллелизма.

Для достижения близкой к пиковой производительности в традиционных ВС от программиста и транслятора фактически требуется соблюдение идеальных условий. На вход каждого из процессоров системы должна поступать последовательность команд, обеспечивающая постоянную полную загрузку функциональных устройств процессора. Как известно, полной загрузки всех ФУ процессора трудно добиться даже в ЭВМ последовательного типа. В параллельных ВС простои неизбежны, во-первых, из-за особенностей алгоритма (см. закон Амдала), во-вторых, из-за сложности балансировки загрузки процессоров. В самом деле, с одной стороны, распределение вычислений и данных по процессорам, равно и как организация информационных обменов и синхронизация, определяется человеком. С другой стороны, продолжительность каждого вычислительного процесса зависит от величин входных обрабатываемых данных, получаемых в процессе вычислений, и не всегда может быть заранее вычислена, что и приводит к неизбежным простоям оборудования и снижению производительности.

Заметим, что традиционные параллельные ВС часто используют экстенсивный путь развития. Производительность отдельных процессоров увеличивается за счет более плотной упаковки транзисторов на кристалле и повышения тактовой частоты, а производительность всей системы в целом — за счет увеличения числа процессоров. Очевидно, ни то, ни другое никоим образом не решает обозначенную проблему простоя оборудования.

В то же время, как показывают исследования, очень многие задачи обладают так называемым **скрытым параллелизмом**, определение схемы которого для программиста-человека является практически неразрешимой задачей. Применение моделей программирования для традиционных архитектур приводит к тому, что скрытый параллелизм задачи не только не выявляется, но даже полностью вырождается. Проиллюстрируем это на следующем примере.

Для описания вычислительного процесса будем использовать так называемый **граф потоков данных (потоковый граф)**. Этот граф состоит из вершин, отображающих операции (сложение, деление, умножение, сравнение и т. п.) и дуг, показывающих потоки данных между теми вершинами графа, которые они соединяют. Пусть некоторый вычислительный процесс может быть представлен графом потока данных, изображенным на рис. 5.31.

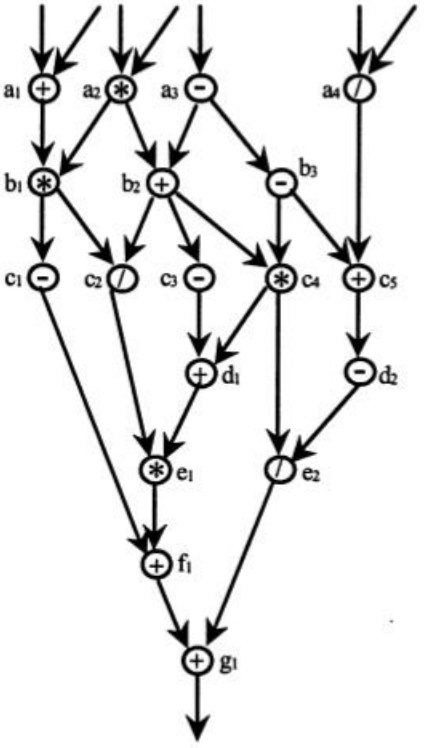


Рис. 5.31. Граф потока данных (потоковый граф)

Граф потока данных на рис. 5.31 фактически можно считать разновидностью ярусно-параллельной формы программы. Вершины графа — арифметические операции — распределены по уровням *a*, *b*, ..., *g*. Каждая вершина имеет свой идентификатор, состоящий из буквы уровня и номера вершины на уровне. На рис. 5.31 хорошо виден скрытый параллелизм алгоритма — вершины-операции одного уровня являются независимыми и могут выполняться параллельно. При выполнении данного вычислительного процесса на ЭВМ традиционной архитектуры мы можем получить примерно следующий порядок выполнения операций (рис. 5.32, показан серой линией).

Очевидно, изображенный порядок выполнения операций сводит на нет весь скрытый параллелизм алгоритма. Теперь представим, что изображенный вычислительный процесс представляет собой одну ветвь параллельной программы. Каждая из ветвей исполняется на отдельном процессоре традиционной архитектуры. Получается парадоксальная ситуация. Сначала программирование алгоритма для фон-неймановской ЭВМ уничтожает его скрытый параллелизм, а затем программистом прилагаются значительные усилия для распараллеливания получившейся программы: выделяются ветви из последовательной программы, распределяются данные по ветвям и организуется взаимодействие ветвей.

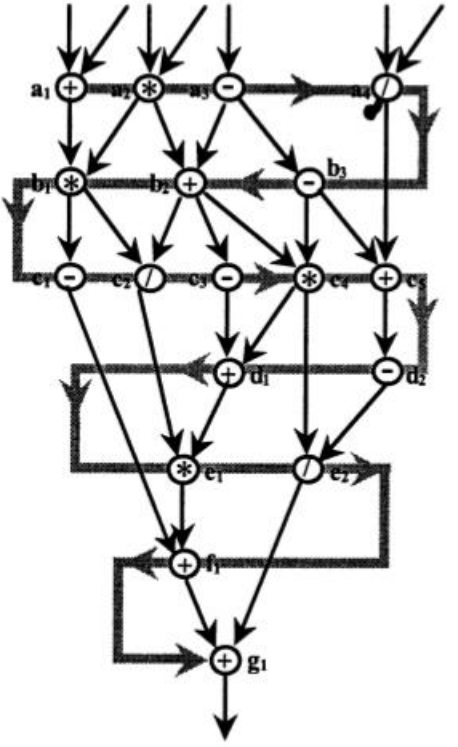


Рис. 5.32. Выполнение алгоритма на традиционной ЭВМ

Стремление устранить указанное противоречие и привело к идее создания ВС, управляемых потоком данных, — **потоковых архитектур**. Основные цели построения подобных систем следующие:

1) человек (программист) должен быть исключен из распределения вычислительных процессов по вычислительным ресурсам системы, а также решения задачи синхронизации параллельных процессов по данным;

2) предельное распараллеливание вычислительных процессов и синхронизация их по данным должны выполняться только на аппаратном уровне в динамике вычислений.

В соответствии с этими целями и был сформулирован основной принцип потоковой обработки — **команда выполняется, когда становятся доступными ее операнды**. ВС, построенные по указанному принципу, называют **потоковыми** или **управляемые потоком данных (dataflow, data driven)** в отличие от традиционных, **управляемых потоком команд**.

В потоковых ВС последовательность выполнения команд задается непосредственно графом потока данных. Вершины-операции выполняются, когда по дугам в вершину поступили все необходимые операнды. Обычно вершина-операция требует одного или двух операндов, а для условных операций необходимо наличие

входного логического значения. При выполнении операции могут быть сформированы один или два результата. Таким образом, у каждой вершины может быть до трех входящих и одна или две выходящих дуги.

При поступлении в вершину всех операндов активируется соответствующая операция, что иногда называют **инициированием вершины**. После выполнения операции результаты передаются по соответствующим дугам к ожидающим их вершинам. Процесс повторяется, пока не будут инициированы все вершины и получен окончательный результат. Очевидно, что одновременно могут быть инициированы несколько вершин, что позволяет автоматически выявить скрытый параллелизм алгоритма.

Выполнение графа потока данных реализует параллельность вычислительных процессов, заданной программой, и исключает конфликтные ситуации по данным. *Фактически в потоковой ВС исключается обычная оперативная память, запись и считывание данных из которой осуществляются операторами присваивания.*

К сожалению, при практической реализации потоковых архитектур возникают сложности, из-за которых на сегодняшний день пока не удалось построить эффективно работающей ВС. *Во-первых*, блок управления потоковой ВС должен быть способен выявлять готовые к исполнению команды. При этом, для того чтобы производительность потоковой ВС превышала или была сопоставима с производительностью традиционной ЭВМ, время поиска готовых команд не должно сильно превосходить времени выборки очередной команды в традиционной ЭВМ. Очевидно, реализовать столь быстрый поиск непросто.

Во-вторых, одновременное исполнение независимых команд потокового графа требует наличия многочисленных параллельно работающих функциональных устройств. Наконец, *в-третьих*, при наличии большого числа ФУ для поддержания высокой эффективности потребуется организация высокоскоростной широкополосной магистрали данных между ФУ и оперативной памятью. Практические попытки разработок потоковых ВС показывают, что одновременное соблюдение всех трех перечисленных условий чрезвычайно затруднено.

Тем не менее теоретические преимущества потоковой обработки перед традиционными архитектурами настолько очевидны, что ученые и инженеры не оставляют попыток практических разработок потоковых ВС. В России подобными исследованиями активно занимается Институт проблем информатики РАН под руководством академика В. С. Бурцева. Коллектив ИПИ РАН предложил настолько удачную архитектуру потоковой ВС, что на ее примере мы проиллюстрируем основные принципы построения систем потоковой обработки.

Рассмотрим фрагмент графа потока данных (рис. 5.33). Операторы программы представлены вершинами графа, каждая вершина имеет уникальный идентификатор (буква и индекс в кружке), однозначно определяющий оператор программы. Идентификатор вершины (a_i, b_i и т. д.) является фактически адресом соответствующего оператора. Данные, поступающие в вершину, являются вход-

ными операндами для соответствующей вершине команды. Каждая команда состоит из кода операции (сложение, деление, умножение и т. п.) и идентификаторов (адресов) вершин, куда необходимо направить результат. Таким образом, результат выполнения любой команды является операндом для одной или нескольких других команд.

Нетрудно видеть, что каждое данное (операнд), передаваемое по дугам потокового графа, содержит **идентификатор (адрес)** той вершины, куда оно направляется. Для того чтобы различить данные, направляющиеся к одной вершине, но относящиеся к разным итерациям цикла или разным подпрограммам, каждому операнду приписывается определенный «**цвет**». Для инициации вершины и выполнения команды необходимо, чтобы в нее поступили операнды одного «цвета». На рис. 5.33 видно, что к вершинам a_2 и a_3 поступили данные от разных итераций (подпрограмм), имеющие разные «цвета».

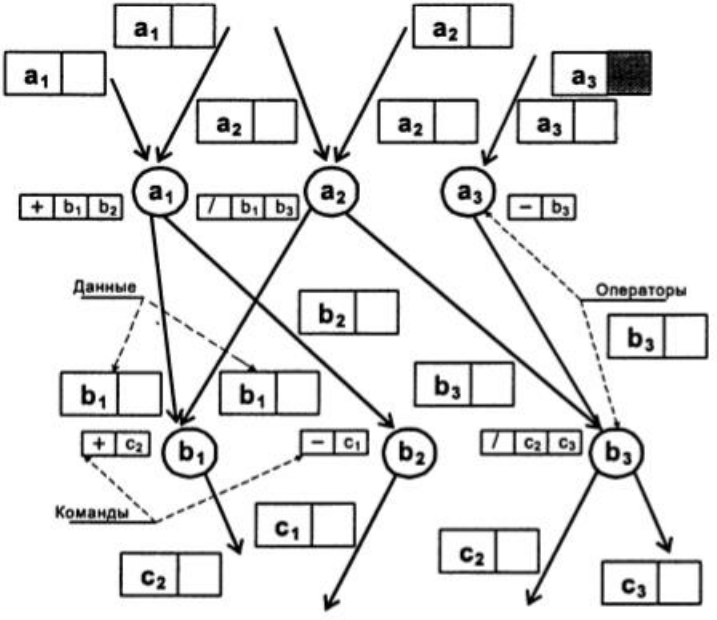


Рис. 5.33. Фрагмент потокового графа

Архитектура потоковой ВС представлена на рис. 5.34. Работа такой системы сводится к следующему. Выходящие из исполнительных устройств данные — результаты выполненных операций — в свою очередь являются операндами для последующих операций. Каждое выходное данное содержит указатель на следующий оператор (вершину потокового графа). **Ассоциативная память** (АП) объединяет данные, относящиеся к одному и тому же оператору (вершине графа). Если парного данного нет в АП, происходит его запись на свободное место АП. Если парное данное есть, то происходит считывание этого данного со стиранием

его в АП и формирование пары данных, направленных к одному и тому же оператору, которая выдается на исполнительное устройство.

Для объединения данных, имеющих единый «цвет» и относящихся к одной и той же вершине потокового графа, АП должна объединять данные по «цвету» и идентификатору вершины потокового графа (адресу). Если пара по этим признакам найдена, АП выполняет ранее описанный алгоритм. В том случае, если парный операнд не находится в памяти, АП производит запись операнда на любое свободное место. Применение ассоциативной памяти в данной архитектуре способствует решению многих проблем, тормозящих развитие принципов управления потоком данных.

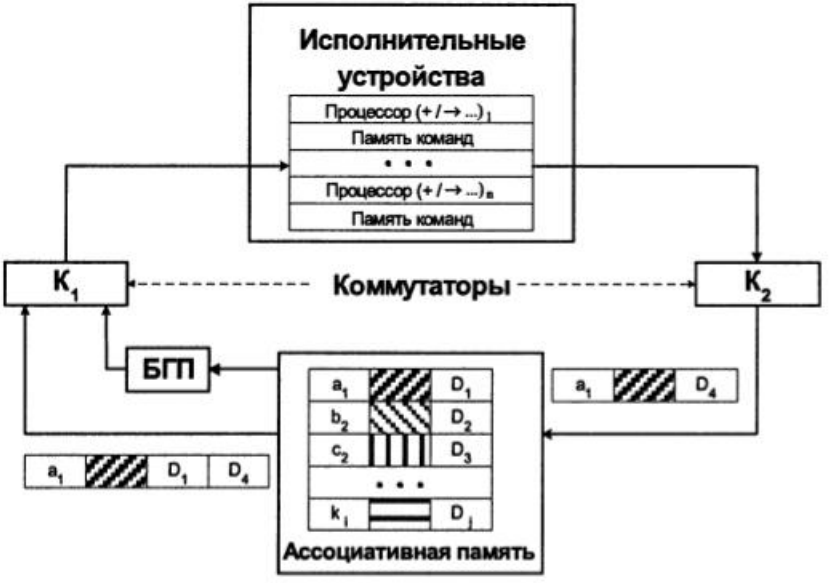


Рис. 5.34. Архитектура ВС, управляемой потоком данных

Фактически АП берет на себя функции управления вычислительным процессом. При этом, в отличие от традиционных ВС, где устройство управления нерегулярно, АП делает вычислительную систему чрезвычайно регулярной структурой, тем самым значительно улучшая технологичность изготовления потокового процессора.

Однако АП является и самым узким местом в вычислительной системе. Дело в том, что скорость работы АП обратно пропорциональна ее объему. Малый объем АП не позволяет организовать массовые вычисления, а большой объем значительно снижает скорость выборки готовых команд.

Для сохранения высокой скорости выборки при больших объемах АП применяется принцип модульного построения, для реализации которого вводятся два **коммутирующих устройства**. Коммутатор K_1 распределяет выходящие из моду-

лей АП готовые команды по свободным исполнительным устройствам. Коммутатор K_2 распределяет выходящие из исполнительных устройств результаты по модулям АП. Если имеются N исполнительных устройств и N модулей АП, то коммутаторы K_1 и K_2 являются коммутаторами $N \times N$. Для эффективного распределения и поиска данных по модулям АП используются специальные хеш-функции. Готовые к исполнению команды поступают в буфер готовых пар (БГП).

Таким образом, к особенностям рассмотренной архитектуры, определяющим ее *преимущества* перед традиционным подходом, относятся следующие.

1. На аппаратном уровне реализуется практически полный параллелизм вычислительных процессов выполняемой задачи, что позволяет достигать очень высокой производительности. Ассоциативная память автоматически определяет все имеющиеся на данный момент времени параллельные процессы и передает их в ИУ на исполнение, одновременно осуществляя синхронизацию по данным.

2. Программист исключен из процесса распределения вычислений, которое осуществляется автоматически в процессе прохождения вычислительного процесса, что обеспечивает высокий коэффициент использования системы.

3. Обеспечивается высокая структурная надежность и технологичность системы. Выход из строя или повреждение нескольких исполнительных устройств или модулей АП не приводит к нарушению функционирования системы. Самое нерегулярное и сложное по технологии устройство управления процессором заменяется регулярной системой АП, устойчивой к технологическим дефектам.

4. Конвейеризация вычислительного процесса за счет работы ассоциативной памяти позволяет нивелировать временные задержки в логических схемах. Из АП на исполнительные устройства поступают не связанные между собой пары данных для исполнения операций. Это позволяет строить систему по конвейерному принципу, исключая временные задержки из вычислительного процесса.

5. АП берет на себя функцию адресной арифметики и целый ряд других функций управленческого характера, существенно упрощая написание программ и повышая производительность процессоров.

6. Потоковый процессор при работе в масштабе реального времени практически не нуждается в традиционном регистре прерываний, реагируя практически мгновенно на сигналы, приходящие в масштабе реального времени. Эта структура не требует затрат процессорного времени на ожидание прерывания. Переход к необходимой программе обработки сигнала, приходящего в масштабе реального времени происходит через ячейку АП, в которую направляется этот сигнал.

В ИПИ РАН был изготовлен макет рассмотренной потоковой ВС, а также были разработаны ее программный эмулятор и язык программирования вместе с транслятором. Эмулятор позволяет запускать оттранслированные программы, оценивать время их исполнения и эффективность использования ресурсов. Результаты проведенных макетных экспериментов выявили основные преимущества потоковой обработки. Во-первых, программы для потоковой ВС получаются очень

компактными (например, решение известной задачи о восьми ферзях реализуется двумя десятками машинных команд). Во-вторых, потоковая ВС, как и ожидалось, позволяет автоматически выявлять скрытый параллелизм задач и эффективно (близко к 100%) использовать вычислительные ресурсы.

В то же время широкому распространению потоковых систем мешает ряд факторов. *Во-первых*, отсутствует отлаженная технология их промышленного производства, существующие макеты не могут рассматриваться даже как предсерийные образцы. *Во-вторых*, потоковые ВС не являются универсальными — с их помощью нельзя решить произвольно взятую задачу, например разработать офисное приложение. *В-третьих*, для использования потоковых ВС необходимо не только полностью переписать программы на новом языке программирования, но и пересмотреть традиционные алгоритмы решения задач для получения возможности их решения на потоковой ВС.

5.9.2. Систолические массивы

Систолические архитектуры (их чаще называют **систолическими массивами — СМ**) представляют собой множество процессоров (в терминологии систолических архитектур они называются **процессорными элементами — ПЭ**), объединенных регулярным образом. Топология объединения может быть любой, хотя чаще всего используются «линейка» и «решетка». Обращение к памяти может осуществляться только через определенные ПЭ на границе массива (рис. 5.35). Выборка операндов из памяти и передача данных по массиву и вычисления осуществляются в одном и том же темпе по тактам синхронизации. Каждый из ПЭ массива «знает», от кого он должен принять входные данные и кому должен отдать результаты на каждом шаге счета. При этом каждый ПЭ на каждом шаге вычислений выполняет небольшую инвариантную последовательность действий.

Рис. 5.35. Обобщенная структура систолического массива

Таким образом, систолический массив — это вычислительная сеть, которая производит ритмические вычисления и передачу данных по системе и обладает следующими характеристиками.

1. **Синхронность.** Данные вычисляются ритмично (при глобальном тактировании) и пропускаются по сети.

2. **Модульность и регулярность.** Массив содержит модульные ПЭ с однородными связями. Вычислительная сеть может неограниченно расширяться.

3. **Пространственная локальность и временная локальность**. Систолические массивы характеризуются локально связанной структурой межсоединений, т. е. пространственной локальностью. В структуре существует распределенная задержка (как минимум единичной длительности), в течение которой могут быть завершены прием, обработка и выдача сигнала от одного ПЭ к другому. Данная задержка определяет временную локальность.

4. **Конвейеризуемость**, определяющая ускорение выполнения, прямо пропорциональна числу ПЭ.

Рассмотрим пример систолического массива, вычисляющего свертку числовой последовательности x_i, $i \in \{0, \ldots, n\}$. Выходная числовая последовательность y_i, $i \in \{0, \ldots, n\}$ определяется формулой

$$y_i = \sum_{j=0}^{n} w_j x_{i-2j}, \tag{5.13}$$

где w_j определяет весовой коэффициент свертки.

Систолический массив для свертки указанного вида будет иметь вид, показанный на рис. 5.36.

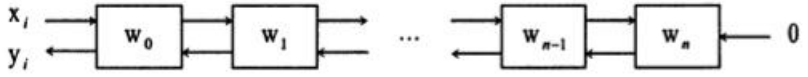
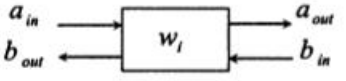


Рис. 5.36. Систолический массив для вычисления свертки

Каждый ПЭ получает две входные величины a_{in} и b_{in}, на каждом такте работы СМ вырабатывает две выходные величины a_{out} и b_{out} по формулам, показанным на рис. 5.37.



$$a_{out} = a_{in}$$
$$b_{out} = b_{in} + a_{in} \cdot w_i$$

Рис. 5.37. Элемент систолического массива для вычисления свертки

На рис. 5.38 показан пример систолического массива для умножения матриц.

Рассмотрим основные свойства систолических архитектур.

1. Простота и регулярность схемы. Систолический массив легко реализуется в виде СБИС или ПЛИС.

2. Параллелизм и связь. Параллелизм определяется алгоритмом, связи — локальные и регулярные, что опять-таки способствует дешевой и простой реализации.

3. Сбалансированность вычислений с вводом-выводом. Следует отметить, что в систолических архитектурах под операциями ввода-вывода понимаются информационные обмены между граничными ПЭ массива и оперативной памятью.

Конструктивно систолические массивы реализуются либо в виде заказных СБИС, либо с помощью ПЛИС в виде дополнительного вычислителя к основной

универсальной ЭВМ, оперативная память которой используется для получения СМ исходных данных и записи результатов. Конечная цель разработки СМ — достижение скорости вычислений, согласованной с шириной полосы ввода-вывода основной ЭВМ.

Систолические массивы используются для аппаратной реализации какого-либо алгоритма. Естественно, что перенастройка уже построенного массива под другой алгоритм невозможна. Однако за счет потери универсальности достигаются высокие показатели производительности на данном алгоритме.

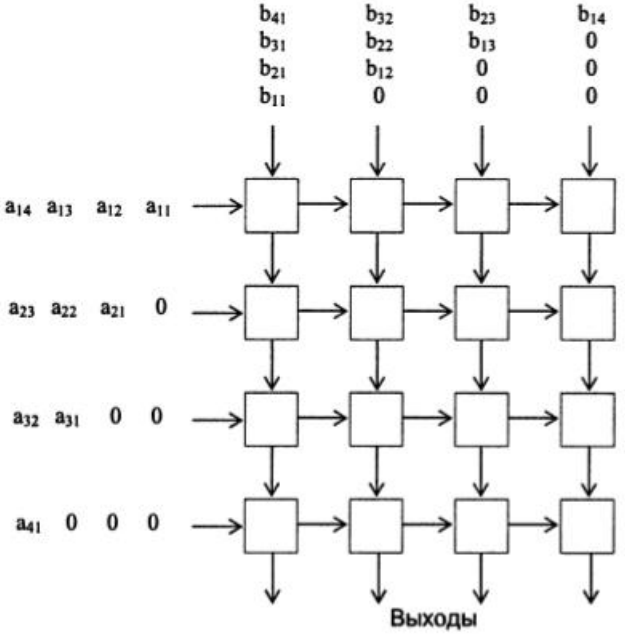


Рис. 5.38. Систолический массив для умножения матриц

Построение систолического массива предполагает проведение **систолизации** — отображение алгоритма в систолический массив. Данное отображение проводится в три этапа.

На первом этапе осуществляется построение **графа зависимостей (ГЗ)**. Рассматривается соответствующее пространственно-временное индексное пространство, в качестве вершин выступают элементарные шаги алгоритма, для отображения зависимости между шагами используются дуги графа.

Второй этап — построение **графа потока сигналов (ГПС)**. Вершины ГЗ, соответствующие шагам алгоритма преобразуются в узлы обработки, зависимости — в ребра связи. В отличие от ГЗ, в ГПС вводятся *задержки*, учитывающие временную локальность будущего систолического массива.

Третьим этапом является собственно **систолизация** — отображение графа потока сигналов в систолический массив.

Существует достаточно широкий класс алгоритмов, чье решение на систолических массивах весьма эффективно и целесообразно. К данным задачам относятся:

- алгебраические операции с матрицами;
- алгоритмы сортировки;
- задачи динамического и математического программирования;
- поиск глобального оптимума сложных математических функций;
- нейронные алгоритмы.

Рассмотрим основные **показатели эффективности** систолического массива, определяющие качество решения им поставленной задачи.

1. Время вычислений T: интервал между началом первого вычисления и окончанием последнего. Время начала первого вычисления отсчитывается с момента поступления на вход массива первой входной величины. Соответственно время окончания вычислений определяется записью последней выходной величины в оперативную память.

2. Конвейерный такт α: интервал между двумя последовательными вычислениями в процессорном элементе. Фактически определяется тактом синхронизации массива.

3. Блочный конвейерный такт β: интервал между началом двух последовательных задач, выполняемых в СМ. При выполнении определенной задачи (например, перемножая две конкретные матрицы) в силу конвейерности вычислений часть ПЭ освобождается раньше других и может начать выполнять следующую задачу (в нашем примере — умножать следующие две матрицы). Отличие блочного конвейерного такта от времени вычислений определяется длиной конвейера.

4. Размер массива — число ПЭ.

5. Число каналов ввода-вывода — фактически число портов доступа к оперативной памяти основной ЭВМ.

Систолические массивы применяются для решения задач трех типов:

1) одной задачи с конечным потоком данных;

2) одной задачи с бесконечным потоком данных;

3) множества задач.

Таким образом, основной принцип систолической разработки можно сформулировать, как *достижение массового параллелизма при минимальных затратах на коммуникации*. При этом применение систолических архитектур ограничено специальным классом алгоритмов, использующих регулярные и локальные структуры данных.

Вопросы для самоконтроля

1. Что такое параллельная обработка информации?
2. Какие уровни параллелизма Вы знаете?
3. Охарактеризуйте типы параллельной обработки.

4. Как записывается ярусно-параллельная форма программы?

5. Выведите закон Амдала и его следствие.

6. Как различает параллельные ВС классификация Флинна?

7. Сравните классификации Флинна и Фенга. Перечислите их достоинства и недостатки.

8. Поясните современную классификацию параллельных ВС.

9. Сопоставьте основные этапы развития параллелизма в архитектуре ЭВМ в мировой и отечественной практике.

10. Что такое схема параллелизма?

11. Какие Вы знаете парадигмы и модели параллельного программирования?

12. Перечислите средства параллельного программирования и дайте им краткую характеристику.

13. Что препятствует векторизации программ?

14. Какие существуют способы устранения зависимостей при векторизации программ?

15. Как вычисляется ускорение при макроконвейерной обработке?

16. Как и за счет чего происходит зацепление функциональных устройств в векторно-конвейерном процессоре?

17. Чем отличаются матричные ВС от векторных?

18. Назовите предпосылки достижения высокой производительности при ассоциативной обработке.

19. Поясните схему ассоциативного процессора.

20. Что собой представляет MIMD-система?

21. Приведите основные характеристики коммуникационных сред MIMD-систем.

22. Как устроен простой коммутатор с пространственным разделением?

23. Как осуществляется построение составного коммутатора?

24. Сравните преимущества и недостатки коммутатора Клоза и дельта-сетей.

25. Постройте коммутатор Клоза размером 20x20.

26. Постройте дельта-сеть размером 27x27.

27. Какими характеристиками обладает граф межмодульных связей распределенного составного коммутатора?

28. Из-за чего могут образовываться дедлоки в коммутаторах?

29. В чем состоит маршрутизация с ациклическим графом зависимостей?

30. В чем состоит маршрутизация по числу пройденных перегонов?

31. Чем отличаются SMP- и NUMA-системы?

32. Назовите подходы и способы поддержания когерентности в ВС с общей памятью.

33. Поясните диаграмму переходов состояний протокола MESI.

34. Как осуществляется запись строки в удаленном ВМ согласно протоколу DASH?

35. Какие архитектурные особенности имели отечественные MPP-системы МВС-100х?

36. Чем отличаются кластерные ВС от массово-параллельных?

37. В чем состоят преимущества и недостатки MPP-систем по сравнению с ВС с общей памятью?

38. Охарактеризуйте направления NOW и PoPC в построении кластерных ВС.

39. Перечислите основные задачи управления вычислительными ресурсами в параллельных ВС. Откуда возникают эти задачи?

40. Что такое «скрытый параллелизм» задачи, и как он нивелируется в ВС традиционной архитектуры?

41. По каким принципам строятся ВС, управляемые потоком данных?

42. Сравните преимущества и недостатки традиционных и потоковых ВС.

43. Перечислите основные свойства и характеристики систолических массивов.

44. Что такое систолизация, и из каких этапов она состоит?

ГЛАВА 6. ПЕРСПЕКТИВНЫЕ НАПРАВЛЕНИЯ В РАЗВИТИИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

6.1. НАНОТЕХНОЛОГИЯ

Существующая технологическая база производства ЭВМ имеет предел своего развития, причем, по прогнозам разработчиков, этот предел будет достигнут в ближайшие годы. Дальнейшие разработки в области построения ЭВМ и ВС направлены на исследования новых физических явлений и создание новых информационных технологий. Прорыв в области создания новых ЭВМ и ВС связывают в первую очередь с *нанотехнологией*. **Нанотехнология** (от греческого «нанос» — карлик) — термин, используемый для описания процессов, происходящих в пространстве, с линейными размерами от 0,1 до 100 нм (1 нм = 10^{-9} м). По мнению одного из создателей водородной бомбы и американской программы СОИ Э. Теллера: «Тот, кто раньше овладеет нанотехнологией, займет ведущее место в техносфере следующего столетия». О роли нанотехнологии говорят такие цифры. Если в 2001 г. оборот мирового рынка нанопродукции составил 45 млрд долларов, то к 2015 г. он достигнет суммы порядка 1 трлн долларов.

Термин **«нанотехнология»** был предложен в 1974 г. японским исследователем Танагучи. Рассмотрим основные этапы развития нанотехнологии в XX в.

1959 г. Лауреат Нобелевской премии Ричард Фейнман заявляет, что в будущем, научившись манипулировать отдельными атомами, человечество сможет синтезировать все, что угодно.

1981 г. Создание Г. Бинигом и Г. Рорером сканирующего туннельного микроскопа — прибора, позволяющего осуществлять воздействие на вещество на атомарном уровне.

1982–1985 гг. Достижение атомарного разрешения.

1986 г. Создание атомно-силового микроскопа, позволяющего, в отличие от туннельного микроскопа, осуществлять взаимодействие с любыми материалами, а не только с проводящими.

1990 г. Манипуляции единичными атомами.

1994 г. Начало применения нанотехнологических методов в промышленности.

История технологии, несомненно, станет яснее, если вспомнить, что сначала за основу для ее построения была взята универсальная метрическая мера — рост человека, который, как правило, лежит в пределах от 1,5 до 2 м. Точное значение не столь уж важно — главное, что мерилом в технологии служит **метр**. Действительно, именно это всегда определяло производство бытовых предметов: ни сами предметы, ни механизмы, применяемые при их изготовлении (от палки и топора до современных станков), неразумно было делать слишком малыми. Можно сказать, что большинство вещей и механизмов — **продукты метровой технологии**.

И тем не менее миниатюризация вещей и механизмов оставалась мечтой умельцев всех времен и народов. Вспомним хотя бы знаменитую блоху нашего умельца Левши.

Переход к **миллиметровой технологии** (1 мм = 10^{-3} м), обусловленный возникновением электроники, произошел в середине XX в. Основные размерные характеристики вакуумной лампы, созданной в 40-х годах (расстояния между проволочками сетки, катодом и анодом и т. д.) не превышали нескольких миллиметров. Однако с развитием электроники потребовалось уменьшить и эти размеры: чем миниатюрнее колба, тем быстрее срабатывает лампа, и тем меньше потребляется энергии. Но лампы приходилось собирать из отдельных деталей, что серьезно осложняло их миниатюризацию.

С последующим сокращением размеров в 1000 раз началась эра **твердотельной микротехнологии**. С ней связан поразительный прогресс вычислительной техники во второй половине XX в. Люди научились размещать 1 млн твердотельных транзисторов в интегральной схеме площадью 1 $см^2$. Кристаллы кремния стали основой интегральных микросхем, миниатюризация которых привела к быстрому росту эффективности вычислительных машин. Рекордные результаты по миниатюризации микросхем на кремнии были достигнуты в 90-х годах благодаря **электронной литографии** — обработке кристаллов электронным лучом.

Несмотря на практически полное отличие микропродуктов и способов их изготовления от изделий метровой и миллиметровой технологий, работают все они на базе одних и тех же классических законов. Например, закон Ома в равной степени справедлив и для бытового электронагревателя, и для интегральной микросхемы. Таким образом, технологии от метровой до микрометровой можно считать классическими.

По мнению ряда специалистов, классические законы уже перестают «работать» при размерах объектов менее 0,5 мкм. Здесь уже действуют законы квантовой физики. Именно здесь и начинается нанотехнология.

Практическая нанотехнология родилась в 1981 г. с изобретением сканирующего туннельного микроскопа. Его изобретателями были Герд Бинниг (Gerd Binnig) и Генрих Рорер (Heinrich Rohrer) из исследовательской лаборатории IBM. Оба ученых за свои труды были удостоены Нобелевской премии. В 1990 г. ученые Айглер и Швейцер из той же лаборатории IBM опубликовали статью «Позиционирование отдельных атомов с помощью сканирующего туннельного микроскопа». В настоящее время Г. Бинниг продолжает работу в фирме IBM и занимается разработкой технологии жестких дисков нового поколения без записывающих/считывающих головок. Сам сканирующий туннельный микроскоп и очень похожий на него сканирующий проектор электронно-лучевой литографии стали прообразами машин будущей нанотехнологии, предвиденной Фейнманом.

В сканирующем туннельном микроскопе очень острая игла движется над поверхностью материала на расстоянии, меньшем величины атома. Через иглу про-

пущен ток, и изменение расстояния между острием «иглы» и поверхностью вызывает резкий «скачок» электрического напряжения. Изменяя величину напряжения, можно «зацепить» атом и переместить его в нужное место. Главная трудность при реализации этой идеи заключается в возможности закрепления атома на новом месте. Для этих целей в настоящее время используют целый ряд методов.

В 1997–1998 гг. агентством DARPA (Defense Advanced Research Projects Agency) США была разработана программа формирования новой наукоемкой технологии, получившей название **молекулярной электроники,** или сокращенно **молетроники** (moletronics). Позже появились и другие термины из этой области, такие, как **бионика, биоинженерия**.

Целью программы DARPA было создание функциональных электронных устройств на основе нанотехнологии. При этом все создаваемые логические элементы и элементы памяти должны функционировать в привычном диапазоне температур, иметь высокую плотность и малую мощность электропотребления. Создаваемая память должна быть энергонезависимой, организована по байтному принципу, и иметь терабайтный объем.

Сами функциональные устройства должны создаваться по принципу направленной самоорганизации, а их структура должна быть трехмерной.

Одним из условий программы было то, чтобы все разрабатываемые устройства в какой-то мере «вписывались» в существующие производства электронной аппаратуры и не требовали его полной реконструкции.

Еще не став практической реальностью, нанотехнология «родила» и новые понятия: *квантовые точки, квантовые диполи, квантовые кучки, квантовые вычисления* (*КВ*), *квантовые компьютеры* (*КК*), *нанотрубки*. Переход к нанотехнологии, несомненно, означает новый этап промышленный революции.

К настоящему времени существует целый ряд разработок, основывающихся на элементах нанотехнологии. Например, компания IBM представила новую технологию хранения информации, с помощью которой можно будет добиться плотности записи порядка триллиона бит на квадратный дюйм. О своем достижении в данной области сообщила и компания HP. В ее лаборатории достигли наивысшей плотности на данный момент и создали демонстрационную 64-битную «микросхему» (скорее, «наносхему») энергонезависимой памяти, в которой роль ячеек памяти играют отдельные молекулы. Этот чип умещается на площади в один квадратный микрон. Кроме того, HP удалось совместить запоминающие и управляющие элементы в одном молекулярном устройстве. По заявлению ее руководителей, у компании HP уже разработана опытная методика производства нанолитографической печати, позволяющая делать копии чипов на пластинах, подобно тому как делаются копии страниц с оригинал-макета в типографиях.

Небольшая компания Nantero сообщила о создании нового экспериментального образца электронной памяти на базе углеродных **нанотрубок**. Инженерам

удалось разместить на кремниевой пластине стандартного размера 10 млрд ячеек памяти, каждая из которых состоит из нескольких нанотрубок. Углеродные нанотрубки (Single-Walled carbon NanoTubes — SWNT) были открыты в 1991 г. ученым С. Ижима (Sumio Iijima, NEC, Япония). Позже было выяснено, что данные образования имеют двойственную природу: они могут себя вести как проводники или как полупроводники, становящиеся проводниками при подаче напряжения определенной величины, при условии закручивания молекулы в спираль.

Нанотрубка — это бесшовный цилиндр, созданный из пленки, сформированной из атомов углерода. Шестиугольные молекулы из атомов углерода получаются из молекул CO (угарный газ) в присутствии катализатора при высокой температуре. Следует отметить, что нанотрубка на порядок прочнее стали и в то же время в 6 раз легче.

Для производства памяти используется стандартный фотолитографический процесс: вначале на подложку из оксида кремния наносится множество нанотрубок, а в ходе дальнейшей обработки неправильно ориентированные трубки удаляются.

Схема памяти представляет собой две пластинки из оксида кремния, расположенные одна на другой на расстоянии около 100 нм. Нанотрубки как бы «подвешены» на верхней пластинке. При подаче на нижнюю пластинку тока трубки меняют свое положение, соединяя две пластинки. Это состояние соответствует наличию в ячейке бита со значением 1. Если же трубка не замыкает пластин, то в ячейке находится бит со значением 0.

Положение нанотрубки определяется влиянием сил Ван дер Ваальса, которые действуют независимо от наличия электропитания. Электрический импульс нужен лишь для изменения положения трубок. При этом на переключение требуется около 0,5 нс против единиц наносекунд у современной оперативной памяти (подобная память получила аббревиатуру **NRAM**). Плотность записи информации в ячейки NRAM постоянно увеличивается и может достичь триллиона бит на квадратный сантиметр, что на несколько порядков больше, чем в случае современной оперативной памяти.

Впрочем, до выхода новой памяти на рынок еще далеко. Углеродные нанотрубки являются все еще экзотическим и дорогостоящим материалом, а производство NRAM, хотя и базируется на традиционной фотолитографии, требует освоения в промышленности. Однако в перспективе NRAM может оказаться востребованным компьютерным рынком для решения проблемы хранения данных.

Следует отметить, что к настоящему времени наиболее значимых результатов в сфере нанотехнологии достигли ученые США, Японии, Финляндии, Голландии и других стран. Старается не отставать в этом направлении и российская наука. На проходившей в 2005 г. в Японии Всемирной выставке «ЭКСПО-2005» одним из наиболее посещаемых был российский павильон «Нанооборудование и на-

нанотехнологии». В частности, большой интерес у посетителей вызвали разработки фирмы **НТ-МТД**. Фирма демонстрировала целый спектр зондовых микроскопов, являющихся инструментом, который позволяет развивать работу в области нанотехнологий.

Успехи нанотехнологии породили целый ряд проектов, связанных с разработкой компьютеров и систем на их основе, использующих физические принципы, отличные от тех, что применялись при создании классических компьютеров. Среди этих проектов можно отметить работы, связанные с созданием молекулярных, оптических, квантовых, криогенных и нейрокомпьютеров.

6.2. МОЛЕКУЛЯРНЫЕ КОМПЬЮТЕРЫ

Вопрос о создании **молекулярного компьютера** встал на повестку дня после того, как было доказано, что биологические и некоторые другие материалы, а также процессы, протекающие в них, можно использовать для создания элементов классической ЭВМ.

Напомним, что критическим элементом полупроводниковой технологии является толщина изолирующего слоя оксида кремния между затвором и проводящим слоем транзистора. Определено, что физическим пределом полупроводниковой технологи является толщина не более 0,015–0,02 микрон (слой в 4–5 молекул). В более тонких слоях начинаются неконтролируемые процессы туннелирования электронов и перегрева, которые нарушают работу транзисторов и компьютера в целом. Кроме того, само формирование интегральной схемы с меньшими размерами транзисторов невозможно на базе стандартных методов фотолитографии.

В таких условиях естественной выглядит идея построения ЭВМ, элементами которой будут выступать молекулы или объединения молекул. Отметим, что толчком к вспышке интереса к молекулярным ЭВМ послужила статья американских исследователей Л. Авирама и М. Ратнера «Молекулярный выпрямитель», опубликованная в 1974 г. Позже сотрудник Военно-морской исследовательской лаборатории (США) Ф. Картер предложил при разработке молекулярной ЭВМ использовать молекулы с резонансной туннельной проводимостью.

При создании молекулярной ЭВМ важным моментом является вопрос о выборе необходимых материалов. Другим не менее важным вопросом является разработка методов для создания будущих элементов молекулярных ЭВМ. Главная трудность при разработке молекулярных ЭВМ— создание **биокристаллов,** или **биочипов (biochip)**, которые будут играть роль элементарных функциональных блоков ЭВМ.

В настоящее время уже выполнен целый ряд работ по выбору белков, пригодных для создания молекулярных логических систем и переключателей. Новые материалы будут иметь малые размеры, низкое сопротивление, небольшое потребление энергии, нерассеиваемую электропроводность. Они будут обладать возможностью сборки в трех измерениях.

В процессе исследований биокристаллов можно выделить три периода.

1. Разработка прототипов, основанных на известных биологических структурах типа мышечных волокон и демонстрация ожидаемых результатов (1985–1987).

2. Накопление знаний, необходимых для создания окончательной технологии построения биокристалла (1988–1992 гг.).

3. Создание прототипа устройства, основанного на молекулярных сборочных узлах с использованием методов, разработанных на предыдущих этапах (1993 г. — настоящее время).

В исследованиях в области создания биокристаллов различные научные школы использовали целый ряд биологических систем, среди которых можно выделить следующие.

1. Аппарат фотосинтеза. В этой системе перенос электронов частично облегчается *хинонами*. **Хиноны** — промежуточные продукты в синтезе красителей, относящиеся к группе циклических **дикетонов**. Трудность заключается в необходимости контроля за нежелательным переносом электронов и поглощением протонов.

2. Микротубулы. Это цилиндрические белковые полимеры, обнаруженные во всех живых клетках. Доказано, что эти полимеры могут играть важную роль в клеточной коммутации и обработке информации.

3. Порфирины, являющиеся компонентами гемоглобина и хлорофилла. В их структуру также входят ионы некоторых металлов. При надлежащей конформации молекул они могут действовать как молекулярная схема с переключением в форме парамагнитного резонанса, вызываемого видимым светом.

4. Видимые пигменты. Система пигментов, определяющая зрительное восприятие, аналогична переключателю в том отношении, что она поглощает фотоны и образует продукты фотосинтеза путем обратимой реакции, при которой эти два состояния могут различаться по спектрам и разности потенциалов.

5. Молекулы ДНК. Используется технология рекомбинантных молекул ДНК для получения цепочек с целью создания трехмерной молекулярной логической схемы.

На настоящий момент программы по созданию элементов молекулярных ЭВМ выполняются как минимум в 12 организациях, включающих национальные агентства, международные компьютерные организации, европейский консорциум, объединяющий фирмы и научные центры многих стран.

В России подобные исследования ведутся в центре фотохимии РАН, Южном научном центре РАН, Институте проблем химической физики РАН, Институте органической химии им. Н. Д. Зелинского, Международном томографическом центре РАН, МГУ, Ростовском госуниверситете и др. По предложению президента

РАН Ю. С. Осипова планируется разработка академической программы в области молекулярных ЭВМ. Реализация этой программы даст шанс России не отстать от мирового уровня.

Хотя о создании молекулярного компьютера говорить пока рано, но основная структурная схема его уже ясна.

Молекулярный электронный прибор состоит из проводящих и непроводящих полимерных цепочек, связанных друг с другом специальным кремниевосодержащим «мостиком». Из этих цепочек формируется переключающее устройство. Переходы электронов из проводящей цепочки в непроводящую переводят устройство из проводящего состояния в непроводящее, и наоборот.

Поскольку биочипы способны избирательно реагировать на различные изменения окружающей среды, они могут отражать не только состояние «включено» и «выключено», но и некоторые промежуточные состояния. Предполагается, что молекулярные электронные приборы могут быть в 1000 раз меньше, а их быстродействие — на несколько порядков больше по сравнению с современными интегральными схемами.

Таким образом, структура молекулярного компьютера должна в общем случае обязательно включать три элемента:

- переключатели;
- память;
- соединяющие проводники.

Рассмотрим последние достижения при разработке этих составляющих.

Одним из вероятных «кандидатов» на роль основы молекулярной ЭВМ является **бактериородопсин (БР)**, открытый в 1971 г. В настоящее время разработана технология создания пленок БР, представляющих собой полимерные матрицы различного состава с включенными в них молекулами белка. Заметим, что впервые в мире такие пленки были получены в нашей стране в рамках проекта «Родопсин» в 80-х годах XX в. Эти пленки получили название «Биохром» и стали пользоваться в качестве фотохромных материалов и среды для голографической записи.

Значительных успехов ученые достигли при работе с молекулами **псевдоротоксана**. Эта молекула, играющая роль переключателя, была изучена еще в 90-е годы, но методы практического ее использования в настоящее время разрабатываются.

При создании образцов молекулярной памяти используют **фульгиды индольного ряда**. Компания Constellation SD объявила о создании первого трехмерного (многослойного) флуоресцентного диска **FMD-ROM**, материалом для которого служат 2-индолилфульгиды и 3-индолилфульгиды. По утверждению фирмы первые готовые к выпуску образцы вмещают на десяти слоях 12-сантиметрового диска до 140 Гбайт данных. Перспективная технология, уже разработанная ком-

панией, позволяет увеличивать число слоев, а значит, и объем записываемой информации.

Японские ученые в качестве объекта исследования используют молекулы **диарилэтена**. Эти работы ведутся в рамках проекта международного научно-технологического центра (МНТЦ), в котором принимают участие Институт органической химии им. Н. Д. Зелинского РАН, Фотохимический центр РАН и НИИ физической и органической химии Ростовского Государственного университета.

Ученые М. Рид (Йельский университет) и Д. Тур (компания HP) создали «сэндвич» примерно из 1000 молекул **дитиола** и поместили его между золотыми электродами. При определенном напряжении, поданном на электроды, этот «сэндвич» удерживает электроны, т. е. хранит данное состояние в памяти, в течение примерно 10 минут. Напомним, что кремниевая память DRAM удерживает такое состояние всего на миллисекунды.

В качестве третьего элемента (проводников) архитектуры молекулярной ЭВМ предлагается использовать: **проводящие полимеры** (допированный полиацителен, политиофен, полианилин и др.); **различные органические проводники**; **нанотрубки** (углеродные и боразатные).

Молекулярные **транзисторы, память** и **проводники** — три составные части будущего молекулярного компьютера, и в их создании по отдельности, как следует из вышеизложенного, есть определенные успехи. Но до решения задачи сборки всех компонентов в работающее устройство еще далеко. Здесь предполагается использовать **принцип молекулярного распознавания**, предполагающий **самосборку и самоорганизацию** сложных ансамблей и агрегатов молекул. Этот же принцип лежит в основе таких сложных структур, как **двойная спираль ДНК, жидкие мембраны** и **глобулярные протеины**.

Задача сборки молекулярной ЭВМ пока не решена. Ученые предполагают делать гибридные устройства, сочетающие достоинства молекулярного подхода с наиболее успешными технологическими вариантами, найденными для кремниевых технологий. Например, используя повышенное сродство атомов серы в органических молекулах к тяжелым металлам (особенно золоту), можно создать контакты между металлическими электродами и молекулярными проводниками.

До сих пор мы рассматривали примеры, когда все функции компонентов компьютера обеспечиваются передвижением электронов в сложных молекулярных ансамблях. Между тем эти функции могут взять на себя и фотоны. Существуют различные варианты фотонных устройств, например, молекулярный фотонный транзистор. В фотонном транзисторе фрагмент молекулы, поглощающий квант света (дипиррилбородифторид), играет роль стокового электрода, следующая молекула (цинковый порфирин) — проводника, а последний излучающий порфириновый фрагмент молекулы соответствует электроду истока. Магниевый порфирин

работает как управляющий электрод-затвор. Если окислить этот затвор, то после поглощения света перенос энергии происходит не на цинковый порфирин, а на неизлучающий магниевый. В компьютерах на подобных транзисторах регулирование всей его работы будет происходить с помощью света.

Интересны исследования заведующего кафедрой «Компьютерные науки» инженерной школы Университета Южной Калифорнии доктора Леонарда Адлемана, которому удалось создать прообраз молекулярного компьютера. В течение 35 лет он проводил исследования по использованию молекулы ДНК в качестве элементной базы компьютера.

По его заявлению, молекулы ДНК в течении четырех часов были использованы для вычисления ответа сложной математической задачи. Микроскопические двойные спирали ДНК рассматривались как путешествующие коммивояжеры. В математике эта задача известна как «проблема ориентированного графа Гамильтона» и посвящена составлению оптимального маршрута. Реально ученым были избраны семь аэропортов, каждый из которых имеет только по одному прилетающему и улетающему рейсу. Необходимо было спланировать маршрут для посещения аэропортов таким образом, что посещение каждого аэропорта осуществлялось лишь один раз. Заметим, что эта задача имеет экспоненциальную сложность.

Однако для реализации предложенной идеи на практике необходимо преодолеть ряд препятствий. Одним из них является то, что пока неясно, как использовать «столь огромное количество дешевых операций для решения реальных вычислительных проблем». Доктор Л. Адлеман говорит: «Я полагаю, что важно суметь применять более тонкие алгоритмы типа таких, что ведут более «интеллигентный» поиск в пространстве решений. Для того, чтобы это произошло, может быть необходимо найти или разработать новые молекулярные «примитивы». Эта задача представляется мне труднейшим препятствием и наиболее интересной с научной точки зрения».

Предполагается, что для проблем, где необходим массивный параллельный поиск, молекулярные компьютеры смогут соперничать с электронными уже в ближайшем будущем.

Касаясь будущего молекулярного компьютера, он говорит: «Сегодняшние суперкомпьютеры могут выполнять около триллиона операций в секунду. Молекулярные компьютеры смогут, предположительно, выполнять более 1000 триллионов операций в секунду. Кроме того, они могут быть в миллиард раз более экономичны энергетически. А размещение информации в ДНК требует около одной триллионной доли пространства, необходимого современным средствам хранения типа видеоленты».

В табл. 6.1 приведены обобщенные характеристики классических и будущих молекулярных ЭВМ.

Таблица 6.1

Сравнительные характеристики классических и молекулярных ЭВМ

Характеристика (параметр)	*Классическая ЭВМ*	*Молекулярная ЭВМ*
Размер транзистора, нм	До 100	1–10
Число транзисторов на 1 см^2	До 10^7	~10^{13}
Время отклика, с	<10^{-9}	До 10^{-15}
Эффективность	Условно принимается равной 1	~10^{11}

6.3. ОПТИЧЕСКИЙ КОМПЬЮТЕР

Под термином «**оптический компьютер**» обычно понимают устройство обработки информации с использованием света.

Говоря об отличительных особенностях света как электромагнитной волны, нужно отметить, что частота световой волны на порядок выше частоты электрических сигналов и волн, используемых в современной компьютерной технике. Потому с помощью световой волны в фиксированный интервал времени можно передавать большее количество информации. Кроме того, поскольку длина световой волны мала, то имеется возможность обработки информации с высокой скоростью.

Следует отметить, что в компьютерах традиционной архитектуры широко используются оптические явления.

В подсистеме ввода-вывода применяются оптические устройства — приводы оптических дисков, сканеры и лазерные принтеры. В подсистеме связи и передачи информации используется оптическое волокно. Передача информации по оптическому волокну заключается в распространении по нему света, при этом скорость передачи на порядок превосходит скорость передачи электрического сигнала. Сочетание оптических и электронных устройств образует **оптоэлектронную** систему.

Первые работы, относящиеся к оптическим или оптоэлектронным ЭВМ, появились в 50-х годах XX в. Интерес к этим ЭВМ был обусловлен целым рядом преимуществ, которыми они обладали по сравнению с традиционными (классическими) ЭВМ. Среди этих преимуществ следует отметить возможность параллельной обработки информации и осуществления сложных двумерных операций типа комплексного преобразования Фурье, корреляции и свертки. Оптические лучи не воздействуют друг на друга, не создают ни коротких замыканий, ни перекрестных помех. Это обеспечивает очень высокую плотность оптических соединений больших массивов оптических логических элементов, недостижимую в системах электронных СБИС. Наиболее эффективно эти ЭВМ можно использовать при обработке изображений, поскольку для таких ЭВМ изображение является естественным двумерным операндом, а основные операции обработки изображений — пре-

образование Фурье и пространственная фильтрация — осуществляются за один такт работы машины. Исходя из этого, сразу же была определена возможная сфера применения таких ЭВМ. Предполагалось, что они будут использоваться в задачах распознавания и идентификации образов, анализа изображений земной поверхности, полученных при аэрофотосъемках, анализе движения облаков и воздушных масс по изображениям, полученным с метеорологических спутников и др.

Исходя из состояния технической базы, первые оптические ЭВМ пытались создавать с использованием таких источников света, как ртутные лампы и даже солнечный свет. Однако такие попытки оказались тщетными, и интерес к оптическим ЭВМ стал затихать. Так, например, компания IBM на создание оптической ЭВМ истратила более 100 млн долларов, но в конце концов прекратила работы в этом направлении.

Вновь интерес к оптическим ЭВМ возник в 60-х годах после появления первых промышленных лазеров. Однако и тогда не удалось создать полностью оптическую цифровую ЭВМ. Были предприняты попытки создать оптоэлектронную **гибридную вычислительную систему (ГВС)**.

Гибридная ВС включала в свой состав **когерентный аналоговый оптический процессор**, выполнявший основные операции по параллельной обработке двумерного потока данных, и **цифровой электронный процессор**, который обеспечивал цифровую обработку, ввод-вывод данных и программное управление процессом обработки.

В оптический процессор информация могла поступать из видеоканала, кинопленки, аналоговой памяти, оперативной или внешней памяти, электронного процессора. Входная информация загружалась в специальное устройство ввода, которое представляло собой матричный пространственный модулятор света. Далее информация поступала в оптическую систему процессора для фильтрации. Операционные фильтры могли быть записаны на обычную фотопленку или оперативный носитель в виде библиотеки стандартных фильтров или же синтезированы в процессе обработки информации. Библиотека стандартных фильтров записывалась в виде двумерной матрицы подобно матрице фурье-голограмм.

Спектр входного сигнала направлялся на нужный операционный фильтр с помощью дефлектора, управлявшего излучением лазера. В спектральной плоскости оптического процессора могло храниться до 10^2 и более операционных фильтров.

Съем информации осуществлялся как в выходной плоскости оптического процессора, так и в ее спектральной плоскости. Для этой цели использовались интегральные фотоматрицы. Информация с выхода этих устройств передавалась на электронный процессор для логического анализа, выработки дальнейших управляющих сигналов и отображения результатов. Результаты обработки могли быть переданы в аналоговую или цифровую память, отображены на дисплее или выведены на печать.

При практической реализации предложенной схемы главная трудность заключалась в отсутствии (на тот момент) эффективных и доступных оперативных устройств ввода информации и синтеза операционных фильтров. Хотя позже и были созданы более современные устройства ввода с оптическим и электрическими входами, ГВС все же не получили широкого распространения. Причиной этого следует считать недостаточный уровень развития технологической базы.

В 1986 г. исследователь из AT&T Д. Миллер создал самый маленький в мире **оптический переключатель** — квадрат со стороной в 10 микрон. Этот переключатель был создан из современных синтетических материалов, обладал скоростью 1 млрд переключений в секунду и при этом не грелся. Миниатюрный переключатель представлял собой зеркальце, на поверхность которого падал лазерный луч. Луч отражался, что соответствовало положению «включено» или 1. Затем на зеркальце падал луч контрольного второго лазера, что делало его неспособным к отражению. Это соответствовало состоянию «выключено» или 0.

Позже были изготовлены симметричные самоэлектрооптические устройства, получившие название **SEED** (SEED — self-electrooptic device), работающие в режиме ИЛИ-НЕ, и симметричные самоэлектрооптические устройства **(S-SEED)**, работающие в режиме защелки (установка-сброс). Из этих устройств можно было изготавливать сумматоры и другие элементы, применяемые для изготовления ЭВМ. Первым практическим применением оптоэлектронных элементов стали системы, которые работали в последовательном режиме и в которых обрабатываемые данные уже имелись в форме оптических сигналов.

В первой оптической ЭВМ в качестве логических элементов использовались соединители (переключатели) оптических каналов на $LiNbO_3$ (ниобат лития), а в качестве элементов памяти — волоконно-оптические линии задержки. Соединители на $LiNbO_3$ использовались в качестве пятиканальных оптических устройств. Схема такого пятиканального переключателя представлена на рис. 6.1.

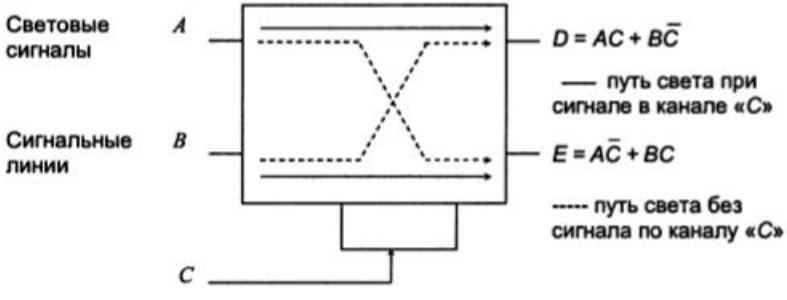


Рис. 6.1. Пятиканальный оптический переключатель

Если в канале *C* сигнала нет, то свет, вошедший по каналу *A*, выйдет из канала *E*, а вошедший по каналу *B* — через канал *D*. При поступлении светового сигнала по каналу *C* комбинация «фотодиод-усилитель» выдает сигнал смещения, переводящий устройство в состояние прямого включения. При этом свет, поступающий

по каналу A, выходит по каналу D, а свет, поступающий по каналу B — по каналу E.

На основе такого переключателя можно построить логические элементы: И, И-НЕ, НЕ, Буфер, ИЛИ и др.

На рис. 6.2 представлен вариант реализации памяти с использованием линии задержки.

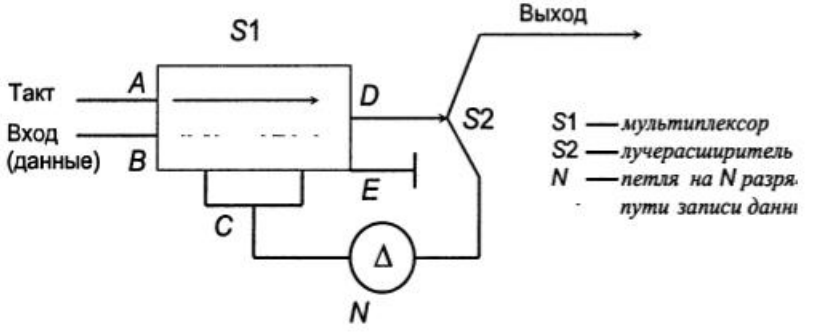


Рис. 6.2. Оптическая память на линии задержки

Известно, что число разрядов, умещающихся в петле, определяется соотношением:

$$N = \frac{L}{L_C}, \tag{6.1}$$

где L — длина световодного контура; L_C — расстояние, проходимое оптическим импульсом за один тактовый период.

Например, для размещения 2 Кбайт в оптической памяти с использованием стеклянного волоконно-оптического световода при тактовой частоте 1 ТГц длина петли должна быть равна 3,3 м. Если тактовая частота будет равна 100 МГц, то длина петли увеличится до 3,3 км. Следует заметить, что верхний предел числа разрядов N, которые можно записать в петле, определяются тепловыми эффектами. Вследствие потерь в оптических волокнах, коммутирующих и входных устройствах, будет происходить падение амплитуды сигнала. Поэтому необходимо предусмотреть компенсацию падения амплитуды.

Одним из способов синхронизации и компенсации падения амплитуды является переключение копии тактового сигнала из канала A в петлю после прихода сигнала на $S1$.

На рис. 6.3 приведена схема оптического 4-разрядного счетчика.

Работу счетчика можно описать следующим образом. Приход импульса на $S1$ переключает его таким образом, что копия тактового сигнала (вход B) направляется через выход E в волоконно-оптический канал. В расщепителе $S4$ импульс расщепляется и, в зависимости от наличия или отсутствия в контуре M циркули-

рующего разряда, направляется или в контур памяти M, или в контур переноса C. Если в контуре M присутствует разряд, то сигнал, поступивший по каналу C переключателя $S2$, вызовет переключение импульса на контур переноса и подавление его в переключателе $S3$. Если в контуре M разряд отсутствует, то импульс подавляется в переключателе $S2$ и заводится в контур памяти переключателем $S3$.

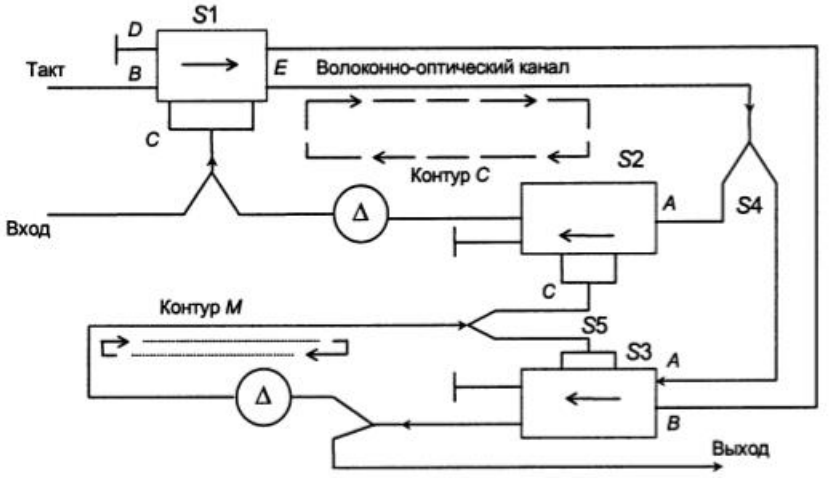


Рис. 6.3 Схема 4-разрядного оптического счетчика: $S1$ — демультиплексор; $S2$ — схема И-НЕ; $S3$ — инверсионная выборка; Δ — линия задержки; C — контур переноса; $S4$, $S5$ — расщепители; M — контур памяти

Важным является вопрос о тактовой частоте, которая зависит не только от частоты повторения лазерных импульсов, но и главным образом от задержек в цепи обратной связи. Применительно к контуру переноса C тактовый период не может быть меньше полной задержки при проходе по контуру, поскольку разряд в контуре задержки должен обойти его и вернуться на переключатель $S1$ в определенный момент времени, с тем чтобы переключить следующий входящий импульс.

Одна из старейших архитектур, разработанных для оптических компьютеров, архитектура OPLA (Optical Programmable Logic Array) — оптическая программируемая логическая матрица. Многолетние работы по реализации этой архитектуры проводились в Японии. На оптических интегральных схемах была реализована вся булева алгебра. Считалось даже, что компьютеры 5-го и 6-го поколений будут реализованы на оптической элементной базе. Но доведение до стадии коммерчески пригодных продуктов оказалось сложней и длительней, чем ожидалось, и в последнее время сообщений о каких-либо успехах в этом направлении не было.

В 1993 г. в Университете Колорадо профессорами Джорданом и Хейрингом был продемонстрирован оптический компьютер размером с небольшой автомобиль и мощностью недорогого ПК. Оперативная память ОЭВМ представляет собой четырехкилометровые петли оптического волокна. По ним циркулируют

импульсы инфракрасного излучения. Четырехметровый импульс кодирует 1, его отсутствие — 0. Кодированные таким образом команды и данные курсируют в линиях задержки, пока управляющий элемент не направит их в процессор. Архитектура получила название *bit-serial architecture* («*последовательная битовая архитектура*»). Потоки информации (лучи) коммутируются в процессоре 66 электрооптическими переключателями на $LiNbO_3$.

По утверждению разработчиков, их детище — просто демонстрация принципиальной возможности создания оптической ЭВМ и не может рассматриваться, как прототип компьютера будущего.

Несмотря на то что простые оптические компоненты (переключатели, счетчики, мультиплексоры, демультиплексоры и т. п.) могут работать на тактовых частотах 40 ГГц и более, энергопотребление многоцветных лазеров значительно превышает даже самые «прожорливые» полупроводники на базе арсенида галлия. Нелишне подчеркнуть, что примитивная схемотехника оптических систем все еще оставляет желать лучшего, делая их малопригодными для реализации вычислительных машин общего назначения.

Работы в области создания оптических ЭВМ велись и в бывшем СССР, ведутся и сейчас в России и на Украине (коллектив ученых АН Украины под руководством М. С. Соскина). Большие успехи были достигнуты в использовании явления нелинейного распространения в оптоволоконных линиях. Еще до создания фирмой AT&T своего оптического переключателя, наши ученые уже использовали в подобного рода переключателях явление низкой нелинейности с удлиненной дистанцией распространения. Это явление было использовано для усиления кумулятивного эффекта взамен применяемого ранее явления высокой нелинейности.

Долгое время в России осуществлялся проект создания оптической сверхвысокопроизводительной вычислительной машины (ОСВМ). Работы по этой теме были начаты еще в 1985 г. в лаборатории академика Г. И. Марчука. В соответствии с проектом оптика в новой ЭВМ должна была использоваться в системах связи и коммутации. Архитектура ОСВМ предусматривала новую организацию вычислительного процесса, структурную надежность и исключение человека из распределения вычислительных ресурсов. ОСВМ должна была дать производительность порядка 10^{15} FLOPS. Однако в отличие от американских разработок эта машина должна была потреблять в десятки раз большую энергию (из-за худших топологических норм) — порядка 10 МВт. Было найдено оригинальное схемотехническое решение, которое исключало задержки распространения сигналов внутри машины и позволило ввести жидкостное охлаждение, раздвинуть блоки и компенсировать высокую рассеиваемую мощность интегральных схем.

В дальнейшем к работе над ОСВМ подключился вновь созданный Институт высокопроизводительных вычислительных систем (ИВВС), директором которого стал академик В. С. Бурцев. К сожалению, в 1998 г. после его ухода работы затормозились.

Первым коммерческим оптическим компьютером считается **Enlight 256** израильской фирмы Lenslet, представленный в 2003 г. Оптический процессор представляет собой цифровой сигнальный процессор с оптическим ускорителем. По сути этот компьютер является гибридным, так как он не полностью оптический, а содержит различные преобразователи света, т. е. ядро компьютера — оптическое, а все остальное — электронное.

Ядро состоит из 256 VCSEL-лазеров, пространственного модулятора света, набора линз и фотоприемников. Производительность процессора составляет 8 TFLOPS, такт — 8 нс. За один такт он может перемножить матрицы размером 256 × 256.

К оптоэлектронному компьютеру второго поколения можно отнести и компьютер специального назначения **Zebra True Recognition** корпорации Mytec Technologies, который был представлен в 1995 г. Этот компьютер предназначен для распознавания дактилоскопических изображений, определения поддельных документов, в том числе страховых полисов, кредитных карточек, а также для идентификации пользователей при входе в корпоративные базы данных. При сравнении отпечатков пальцев он работает сразу со всей картинкой, а не с ее фрагментами, как это делают современные системы.

В заключение заметим, что ближайшее будущее компьютерной техники — за машинами, в которых будут творчески сочетаться компоненты оптики и электроники. Электронные узлы процессора могут быть связаны высокоскоростными световыми волнами информации, которые уже не будут причиной перегрева медных проводов. Электроника хорошо зарекомендовала себя в вычислительных операциях, оптика — в передаче информации. Так что завтрашним днем можно считать оптикоэлектронный компьютерный симбиоз.

6.4. КВАНТОВЫЕ КОМПЬЮТЕРЫ

Среди проектов, посвященных созданию принципиально новых ЭВМ, наиболее перспективным представляется разработка квантового компьютера. **Квантовый компьютер** будет состоять из компонентов субатомного размера и работать по принципам квантовой механики.

Работы в этом направлении ведутся во многих странах мира, но лидирующее положение занимают США. В США работами в этом направлении занимаются такие ведомства, как **NSF** (National Science Foundation), **NIST** (National Institute of Scince and Technology), **DoD** (Departament of Defense), **DARPA** (Defense Advanced Research Projects Agency), **ONR** (Office of Naval Research) и др.

Кроме того, этими же вопросами занимаются и фирмы — производители ЭВМ, в первую очередь IBM. Одним из ведущих мировых центров в этой сфере является Центр спинтроники и квантовых вычислений (CSQC) Калифорнийского университета в Санта-Барбаре.

Ведутся подобные работы и в России. В частности, этими вопросами занимаются институты РАН — Теоретической физики им. Л. Д. Ландау, Физико-технологический и др.

Следует отметить, что уже и в настоящее время существуют множество систем, в которых квантовые эффекты играют существенную роль. В качестве примеров таких систем можно привести различные виды квантовых генераторов (лазеры, мазеры), современные микросхемы, диоды Ганна и др.

Принято считать, что начало работам по квантовым компьютерам положила опубликованная в 1982 г. статья Р. Фейнмана, в которой нобелевский лауреат в области физики обратил внимание на то, что если имеется некоторое **квантовое устройство**, т. е. подчиняющееся законам квантовой механики, то его возможности будут значительно больше возможностей классических компьютеров.

Справедливости ради следует отметить, что еще в 1980 г. советский математик Ю. И. Манин в своих статьях «Вычислимое и невычислимое» и «Доказуемое и недоказуемое» описал сюжет про **квантовые автоматы**, где он говорит о некоторых кардинальных отличиях этих автоматов от классических.

В середине 80-х годов XX в. появились работы Д. Дойча, Е. Бернстайна и У. Вазирини, А. Яо, в которых были описаны модели квантового компьютера. Например, была описана модель квантовой машины Тьюринга. Появившаяся в 1994 г. статья П. Шора (P. W. Shor, сотрудник фирмы Lucent Technologies, математик) вызвала лавинообразный поток публикаций о **квантовых вычислениях (КВ)**. П. Шор предложил квантовый алгоритм, позволяющий производить быструю факторизацию больших чисел. Напомним, что все известные алгоритмы для обычного компьютера — экспоненциальные, т. е. время их вычисления растет как экспонента от числа знаков в записи факторизуемого числа. Так, например, факторизация 129-разрядного числа потребовала бы 500 MIPS-лет или 8 месяцев непрерывной работы системы из 1600 рабочих станций, объединенных через Интернет. А факторизация числа в 500 разрядов потребует времени, превышающего возраст Вселенной, и одновременную работу всех существующих в мире компьютеров. Алгоритм П. Шора, по сравнению с другими, дает многократное ускорение вычислений, причем, чем длиннее факторизуемое число, тем значительнее выигрыш в скорости.

В 1996 г. коллега П. Шора Л. Гровер (L. Grover) предложил квантовый алгоритм быстрого поиска в неупорядоченной базе данных. Этот алгоритм позволяет не только ускорить процесс поиска, но и увеличить примерно в 2 раза число параметров, учитываемых при поиске оптимальным способом.

Независимо от П. Шора наш ученый А. Китаев (ИТФ им. Л. Д. Ландау) также разработал квантовый алгоритм для решения задачи о дискретном логарифме и некоторых других более общих задач.

Именно середину 90-х годов XX в. можно считать временем рождения новой отрасли вычислений — **квантовых вычислений (КВ)**, т. е. вычисления на **квантовом компьютере (КК)**.

Хотя самого КК еще не существует, но уже имеются подходы для его построения.

Прежде чем рассматривать структуры КК, кратко рассмотрим суть КВ. В КВ, как и в обычных компьютерах, оперируют понятием **бит**, которое в этом случае носит название **кубит (qubit, quantum bit)**. Если обычный **бит** — это пространство состояний из двух элементов 0 и 1, то **кубит** — это двумерное комплексное пространство. В квантовой системе можно выполнять только унитарные преобразования пространства состояний системы. С точки зрения геометрии такие преобразования — прямой аналог вращений и симметрий обычного трехмерного пространства. Согласно принципу суперпозиции, можно складывать состояния, вычитать их и умножать на комплексное число.

В самом общем случае элементарным шагом при КВ является унитарная операция ***L*-кубитовой суперпозиции** состояний регистра из L кубитов. Заметим, что при этом выполняется параллельная обработка сразу всех 2^L комплексных амплитуд. Для классической ЭВМ подобная операция потребовала бы 2^L отдельных элементарных шагов. В КВ вычислительный процесс носит характер интерференции, т. е. комплексные амплитуды состояний многих кубитов могут складываться конструктивно и деструктивно.

Кубит — квантовая частица, имеющая два базовых состояния. Этим состояниям могут соответствовать, например, основное и возбужденное состояние атома, направления вверх и вниз спина атомного ядра, два возможных положения электрона и др.

Вне зависимости от принятой структурной схемы основным элементом КК является **квантовый регистр**. Квантовый регистр — это цепочка кубитов, над которыми можно выполнять одно- и двухбитовые логические операции. К базовым состояниям квантового регистра, образованного L кубитами, так же как и в классическом регистре, относятся все возможные последовательности нулей и единиц длины L. Всего возможны 2^L различных комбинаций. Их можно считать записью чисел в двоичной форме от 0 до $2^L - 1$. Но в отличие от классического регистра, кроме этого, существуют еще и состояния суперпозиции, задаваемые комплексными амплитудами.

Следует заметить, что небольшие квантовые регистры ($L < 20$) могут служить лишь для демонстрации отдельных узлов и принципов работы КК, практической ценности они не имеют. Практически ценный КК должен содержать не менее тысячи кубитов.

Независимо от физической основы КК, принципиальная схема его работы может быть в упрощенном виде описана следующим образом.

До ввода информации в КК все кубиты квантового регистра должны быть приведены в основные **базисные** (булевые) состояния. Эта операция называется **инициализацией**. Далее каждый кубит подвергается селективному воздействию, например с помощью импульсов внешнего электромагнитного поля, управляемого классическим компьютером. С помощью этого поля кубиты будут переведены в неосновное состояние, что приведет состояние всего квантового регистра в суперпозицию базисных состояний, задающих бинарное представление числа n в виде:

$$n = \sum_{i=1}^{N} n_i \cdot 2^i. \qquad (6.2)$$

На рис. 6.4 представлена структурная схема квантового компьютера. При выборе конкретной схемы КК необходимо решить три принципиальных вопроса.

1. Выбор **элементной базы** (физической среды), которая будет обеспечивать необходимое количество кубитов.

2. Выбор **физического механизма**, определяющего взаимодействие между кубитами.

3. Определение способа **управления кубитами** и **измерения их состояния** на выходе.

В настоящее время ведутся исследования над двумя различными архитектурами КК: типа клеточного автомата и в виде сети логических элементов. Принципиальных преимуществ ни одна из этих архитектур не дает.

Рис. 6.4. Структурная схема квантового компьютера

В качестве **физических явлений**, используемых основой для логических элементов КК, предлагается использовать: взаимодействие одиночных поляризованных фотонов или лазерного излучения с веществом или отдельными атомами; квантовые точки; ядерный магнитный резонанс (ЯМР); объемный спиновой резонанс.

Наиболее реальной на сегодняшний день является реализация КК, использующего объемный спиновой резонанс. В качестве кубитов здесь используются ориентации ядерных спинов. Работа логических ячеек и запись кубитов осуществляются радиочастотными электромагнитными импульсами со специально подо-

бранными частотой и формой. Основным логическим элементом такого КК будет управляемый инвертор. Ожидается, что уже в ближайшем будущем будет создан КК с разрядностью в 10 кубит.

К настоящему времени определены следующие основные требования, предъявляемые к физической среде, в которой будет реализован КК.

1. Физическая система, представляющая полномасштабный КК, должна состоять из точно известного числа кубитов, число которых должно быть не менее 10^3.

2. Должна быть обеспечена возможность приведения системы в известное начальное состояние (инициализация входного регистра).

3. Необходимо обеспечить высокую степень изоляции КК от внешней среды. Система кубитов должна быть слабо связна с окружением.

4. Необходимо уметь изменять состояние системы в соответствии с заданной последовательностью унитарных преобразований ее фазового пространства.

5. Необходимо обеспечить с достаточно высокой степенью надежности измерение состояний системы на выходе.

Наиболее сложными при реализации являются требования 3 и 4. Рассмотрим в практическом плане те проекты создания КК, которые реализуются в настоящее время.

1. Первый прототип КК, основанный на взаимодействии между заряженными ионами и управлении ими с помощью лазера ИК-диапазона, был предложен австрийскими физиками И. Цираком и П. Цоллером в 1995 г. Реализация этого проекта показала основные недостатки используемого метода: необходимость создания сверхнизких температур, трудности в обеспечении устойчивости состояний ионов в цепочке и ограниченность возможного числа кубитов значением $L \leq 40$. Тем не менее работы в этом направлении ведутся в Лос-Аламосской лаборатории (LANL) и Национальном институте (NIST) США.

2. В 1997 г. в Массачусетском Технологическом институте, лаборатории LANL (США) и Оксфорде (Великобритания) были выполнены первые эксперименты по использованию в качестве кубитов атомных ядер с ядерными спинами $I = 1/2$. Первоначально для экспериментов использовались молекулы 2, 3-дибромотиофена и трихлорэтилена. Позднее использовались молекулы цитозина, хлороформа, аланина и других жидкостей с числом спинов-кубитов $L = 3, 5, 6, 7$. Управление процессами осуществлялось с помощью радиочастотных импульсов. Компьютер такого рода получил название **ансамблевого КК** на основе ЯМР.

3. В 1998 г. Д. В. Аверин предложил использовать в качестве кубитов зарядовые состояния куперовских пар в квантовых точках, связанных переходами Джозефсона. В 1999 г. в Японии (фирма NEC) на этих принципах был создан первый твердотельный кубит.

4. Российский исследователь М. В. Фейгельман (ИТФ им. Л. Д. Ландау) предлагает собирать квантовые регистры из миниатюрных сверхпроводящих колец.

Каждое кольцо выполняет роль кубита, а состояния 0 или 1 соответствуют направлению электрического тока в кольце. Переключение кубитов можно осуществлять с помощью магнитного поля.

5. Группа, возглавляемая американским ученым И. Чангом, объявила о сборке 5-битового КК, где в качестве кубитов используются спины ядер некоторых органических молекул.

6. Группа ученых из фирмы IBM и Станфордского университета продемонстрировала семикубитный КК для факторизации чисел по алгоритму Шора. Компьютер правильно определил, что делителями числа 15 являются числа 5 и 3. По состоянию на 2001 г. это было самое сложное вычисление, выполненное КК. Компьютер представляет собой пробирку с 10^{18} молекул, имеющих 7 ядерных спинов. «Программирование» такого КК осуществляется с помощью электромагнитных импульсов разной частоты. Для получения результата используется ЯМР-сканер.

7. В 2004 г. в Институте компьютерной архитектуры и технологий программирования имени Фраунгофера разработан первый онлайновый эмулятор КК. Установка эмулирует работу 31 кубита и используется для проверки квантовых алгоритмов. Эмулятор представляет собой кластер из 32 узлов на базе процессоров AMD Athlon 3200 с общим объемом ОП 56 Гбайт.

8. В 2005 г. европейское отделение компании Hitachi объявило о создании «базисного элемента» КК. Работы проводились совместно с Кембриджским университетом. Полученный элемент представляет собой микроскопическую частицу размером 50 × 150 нм. Для работы элемента необходима температура, близкая к абсолютному нулю (–273°C). Полученное состояние кубитов, позволяющее выполнять параллельные вычисления, длилось 200 нс, что в 100 раз больше предыдущих результатов. Для сборки полноценного КК необходимо иметь сеть из 10 тыс. таких элементов.

Предполагается, что к 2010 г. появятся первые КК с $L = 100$ кубитов.

6.5. КРИОГЕННЫЕ ЭВМ

Под **криогенной ЭВМ** понимается такой компьютер, элементная база которого основана на явлениях сверхпроводимости и использовании криогенной техники. Понятно, что выделение таких ЭВМ в отдельный класс несколько условно, так как явление сверхпроводимости используется и при разработке других классов и типов ЭВМ. Однако в учебных целях мы считаем целесообразным рассмотреть этот класс ЭВМ отдельно.

Следует отметить, что в большинстве разработок, посвященных криогенным ЭВМ, используется **эффект Джозефсона**, предсказанный в 1962 г. Эффект связан с протеканием сверхпроводящего тока через тонкий слой диэлектрика, разделяющего два сверхпроводника (так называемый контакт Джозефсона).

Явление сверхпроводимости, т. е. резкое падение сопротивления материала при охлаждении до температур, близких к абсолютному нулю, было открыто в

1911 г. датским физиком Оннесом. Идея создания ЭВМ, элементная база которой использует явление сверхпроводимости, была высказана еще в 1948 г. Предлагалось использовать сверхпроводящее кольцо (СП), которое могло иметь два устойчивых состояния: наличие циркулирующего незатухающего тока можно было кодировать как 1, а отсутствие — как 0. В 1956 г. была предложена конструкция первого СП-ключа — **криотрона**. Фирма IBM первой изготовила микросхему на пленочных криотронах, но в дальнейшем эту работу прекратила. Ряд фирм тоже предпринимали попытки создания сверхпроводящей (криогенной) ЭВМ, но также вынуждены были прекращать работы из-за их сложности. Та же фирма IBM в 1965 г. возобновила работы в этой области, но в 1983 г. работы опять были прекращены. В 1986 г., когда ее сотрудники К. Мюллер и Дж. Беднорц открыли высокотемпературную проводимость, работы вновь были возобновлены.

Рассмотрим основные причины, которые заставляют исследователей заниматься криогенной ЭВМ. Хорошо известно, что быстродействие ЭВМ определяется в первую очередь временем срабатывания ее элементов. Независимо от поколения ЭВМ до сих пор цикл ЭВМ был примерно в 50 раз больше времени срабатывания отдельного элемента (вентиля). Если предположить, что это соотношение сохранится и дальше, то для достижения цикла 1 нс необходимо иметь элементы с временем срабатывания 20 пс.

Если в качестве элементной базы ЭВМ использовать полупроводники, то такое быстродействие можно получить, лишь уменьшив длину канала до субмикронных размеров. Этого в принципе можно достичь, используя новые полупроводниковые материалы с высокой подвижностью носителей, охлаждаемые, например, жидким азотом. Однако для получения цикла 1 нс недостаточно наличия сверхбыстродействующих элементов, их необходимо еще так упаковать, чтобы наиболее длинный путь сигнал проходил не более, чем за 1 нс, а это в лучшем случае 10 см. Из этого следует, что ЭВМ с циклом в 1 нс должна иметь объем в несколько кубических дециметров.

Естественно, что в таком объеме трудно сосредоточить многокиловаттную мощность, которую потребляет современная высокопроизводительная ЭВМ, содержащая 10^7–10^8 транзисторов на процессор. Поэтому возникает новая проблема — уменьшение потребляемой мощности. Вот по этому показателю СП-элементы и находятся вне конкуренции. Благодаря их охлаждению до температуры 4,2° К, при которой тепловые потери незначительны, уровень информационных сигналов у них составляет 1 мВ по сравнению с 1 В у транзисторов. Поэтому, если у биполярных транзисторов энергопотребление составляет милливатт на вентиль, у полевых транзисторов — сотни микроватт, то у СП-элементов — единицы микроватт.

Серьезная проблема, возникающая в цифровой полупроводниковой технике, — это межэлементные соединения на кристалле. Линии связи на кристалле

определяют все основные параметры и в особенности быстродействие, которое с повышением степени интеграции снижается за счет увеличения средней длины, площади и погонного сопротивления металлизированных проводников. В современных ЭВМ 40% времени цикла теряется на соединениях. Эту проблему также можно решить с использованием СП-элементов, так как СП-полосковая линия является идеальным средством передачи сигналов без искажения и затухания до частоты 10^{12} Гц. Задержка в такой линии составляет 0,08 нс/см, что значительно меньше, чем в обычной линии. Учитывая, что токонесущие возможности СП-линий выше, чем у обычных линий, эти линии можно делать более узкими и размещать более плотно и в меньшем количестве слоев.

Если ориентироваться на уже существующие принципы организации вычислительного процесса, то согласно принципу Ландауэра, при работе в рамках классической логики любое переключение транзистора приводит к выделению тепла, пропорционального температуре транзистора. Если понизить температуру транзистора, можно будет понизить напряжение питания и, следовательно, уменьшить тепловыделение (снижать напряжение питания без снижения температуры процессора нельзя, так как это приведет к сбоям в работе).

Как сильно требуется охладить процессор, чтобы добиться существенного выигрыша в тепловыделении? Из основного соотношения Ландауэра видно, что охлаждение процессора даже до температуры жидкого азота (77,4° К) не дает больших преимуществ, так как снижает тепловыделение по сравнению с режимом работы при комнатной температуре всего лишь в 4 раза. Если процессор без охлаждения рассеивал, допустим, мощность 60 Вт, то при температуре жидкого азота он будет рассеивать мощность 15 Вт. Охлаждение до температуры жидкого гелия (4,2° К) понижает температуру вычислительного процесса примерно в 100 раз, что дает для рассматриваемого случая мощность рассеяния 600 мВт.

Производительность нанокомпьютера, охлаждаемого жидким гелием, можно оценить следующим образом. Теплота испарения жидкого гелия примерно равна $3 \cdot 10^3$ Дж/л. Таким образом, одного кубического миллиметра жидкого гелия, расходуемого за 1 секунду при температуре 4,2° К, будет достаточно для отвода ландауэровского тепла от машины с вычислительной производительностью примерно $5 \cdot 10^{19}$ MIPS. Если предположить, что одновременно будут переключаться 100 млн одноэлектронных транзисторов, то рабочая частота нанокомпьютера может быть выше 100 ГГц, а тепловыделение — лишь 3 мВт. Создание криогенного наночипа — дело вполне реальное, так как в системе замкнутого оборота криогенного кулера должно быть всего несколько кубических миллиметров жидкого гелия.

При широком коммерческом производстве гелиевые кулеры для ПК будут размером не более воздушных вентиляторов для современных процессоров. При этом они должны будут отводить тепловую мощность всего несколько милливатт.

Для суперкомпьютерных центров можно создать более мощные нанокомпьютеры со стационарными криогенными установками.

С другой стороны, 3° К — это температура космоса. Почему бы космос с его неограниченными холодильными ресурсами не подключить к решению проблемы роста вычислительных ресурсов на Земле? Суперкомпьютерные центры, расположенные на геостационарных орбитах с дешевым космическим холодом, оснащенные мощными информационными каналами связи с Землей, — новое направление развития IT-бизнеса в будущем.

Хотя температуру рабочей среды компьютера можно еще понижать по сравнению с температурой жидкого гелия, при этом будут быстро расти и затраты на охлаждение.

Известны и другие физические механизмы, используя которые, можно оптимизировать термодинамику классического компьютера. Дело в том, что принцип Ландауэра выводится в предположении, что вычислительная среда характеризуется одной температурой *T*. Однако в физике известны среды с двумя и более температурами, т. е. являющиеся термодинамически неравновесными. Пример — всем хорошо известные газоразрядные лампы дневного света. Атомно-молекулярная подсистема здесь характеризуется комнатной температурой (300° К), а система свободных электронов — температурой в 30–50 раз большей (10000° К). Поэтому в вычислительной среде можно создавать переохлажденную рабочую подсистему с очень низкой температурой, а по завершении вычислительного процесса считывать результат еще до того, как начнут сказываться потери информации в результате возвращения системы к состоянию теплового равновесия. Последовательная реализация такого подхода приводит к идее оптимального сочетания возможностей квантового и классического компьютеров. Например, можно использовать взаимодействие холодных квантовых пучков легких частиц с массивами более теплых тяжелых частиц.

Таким образом, у СП-элементов имеется одновременное сочетание таких возможностей, как сверхвысокое быстродействие, низкое энергопотребление и идеальные характеристики соединений. Однако несмотря на все преимущества СП-элементов, реальной криогенной ЭВМ пока нет. Разработаны лишь отдельные микросхемы, узлы. Имеются отдельные проекты создания ЭВМ. Рассмотрим наиболее оригинальные из подобного рода разработок.

Фирмой IBM на джозефсоновских элементах спроектирован гипотетический процессор с архитектурой полупроводниковой ЭВМ IBM-3033. Если бы этот процессор был реализован, то производительность ЭВМ возросла бы с 4,9 до 70 млн оп/с, а суммарное потребление мощности и занимаемая площадь уменьшились бы соответственно с 190 кВт и 70 м^2 до 15 кВт и 16 м^2. Этой же фирмой в 1986 г. была предпринята попытка создать макет супер-ЭВМ со следующими характеристиками: производительность от 70 до 250 млн оп/с; емкость ОП до

16 Мбайт; размеры ЭВМ 15 × 15 × 5 см; потребляемая при температуре 4,2° К мощность 12,5 Вт. Для обеспечения этой мощности необходимо было иметь рефрижератор мощностью 30 КВт и стоимостью 30 тыс.долларов. Этот макет разрабатывали 115 исследователей, расходуя ежегодно до 20 млн долларов.

Однако поставленная задача не была решена, так как не удалось создать сверхбыстродействующее ЗУ емкостью 256 Кбайт для связи процессора с внешним ЗУ; ошибочно были выбраны материалы (легкоплавкие полупроводники), из-за чего не удалось создать надежные, долговечные, выдерживающие многократное термоциклирование туннельные джозефсоновские контакты; не была решена проблема воспроизводимости параметров элементов; не были отработаны вопросы по сборке и отладке ЭВМ.

Целый ряд фирм Японии (Hitachi, NEC, Fujitsu), а также ряд институтов и университетов занимались разработкой криогенной ЭВМ. В 1981 г. в Японии была принята программа по созданию в течение 8 лет ЭВМ с производительностью 100 MFLOPS в однопроцессорном варианте и 10 GFLOPS при максимальной конфигурации. При создании ЭВМ были использованы две альтернативные технологии: на охлаждаемых до –200° С элементах из арсенида галлия и на джозефсоновских элементах. В рамках этой программы японцам удалось решить проблему стабильности и устойчивости к многократному термоциклированию (более 1082 термоциклов, 4,2° К, –300° С), повысить качество, однородность и воспроизводимость характеристик, а также быстродействие джозефсоновских туннельных контактов (ТК) благодаря переходу от свинцовой технологии к ниобиевой.

Фирмой Fujitsu были разработаны логические элементы с задержкой 1,5 пс, и на их основе создана микросхема четырехразрядного микропроцессора. Микросхема имеет размер 5 × 5 мм и содержит 1841 логический вентиль. Подобные разработки были выполнены и другими фирмами. В целом можно сказать, что японскими фирмами были продемонстрированы рекордные характеристики по быстродействию и потребляемой мощности, но создать высокопроизводительную криогенную ЭВМ пока не удалось.

В связи с открытием высокотемпературной проводимости (ВСТП) было высказано немало прогнозов о том, что теперь уже не будет препятствий для создания криогенной супер-ЭВМ, поскольку жидкий азот доступнее, чем жидкий гелий. На ВСТП возлагаются большие надежды. Об этом говорят те ассигнования, которые выделяются на НИР в этой области. В США за 1988–1990 гг. они составили 548 млн долларов, в Японии в 1989 г. было выделено 88,9 млн долларов. По американским прогнозам, мировой рынок ВСТП-продукции к 2010 г. составит более 2 млрд долларов, а по японским прогнозам он возрастет до 10 млрд долларов. Однако при практической реализации идей по использованию ВСТП нужны новые подходы для построения цифровых элементов. Поэтому появления криогенных ЭВМ на рынке СВТ следует ожидать не ранее чем через 10 лет.

6.6. НЕЙРОКОМПЬЮТЕРЫ

6.6.1. Общие положения

Прежде чем рассматривать принципы построения нейрокомпьютеров, вспомним принцип программного управления, используемый во всех ЭВМ с архитектурой фон Неймана. Согласно этому принципу любая решаемая задача должна быть вначале представлена в виде программы — жесткого алгоритма, учитывающего все возможные ситуации и события. Однако на практике приходится решать задачи, для которых нелегко либо практически невозможно разработать единый алгоритм их решения (так называемые трудно формализуемые и неформализуемые задачи). Это, например, такие задачи, как задача распознавания образов и изображений, звука и т. д.

В то же время в ряде случаев человек решает эти задачи гораздо эффективнее ЭВМ (например, узнавание человека в толпе только по его промелькнувшему лицу). Это привело ученых-кибернетиков к необходимости изучения механизмов функционирования нейросетей (нервных систем) животных и человека, чтобы разработать их математические модели с заданной степенью приближения. На основе разработанных моделей можно программно сымитировать обучение и поведение нейросетей и (или) построить на приемлемом технологическом базисе вычислительные устройства, реализующие их функции — **нейрокомпьютеры**. Как уже упоминалось, наука, занимающаяся этими проблемами, получила наименование биокибернетика, бионика или биомиметика.

Стоит, однако, отметить следующее. Во-первых, в нейросетевом логическом базисе возможно более эффективное решение задач, не относящихся к трудноформализуемым. Это, например, задачи с ярко выраженным естественным параллелизмом обработки сигналов и изображений. Во-вторых, помимо всего вышесказанного, фон-неймановским машинам присущи следующие недостатки, от которых свободны нейрокомпьютеры:

- необходима обширная и тщательная проработка алгоритмов решаемых задач;
- искаженные исходные данные и даже малая их часть приводят к получению неверных результатов или к сбою в работе машины;
- случайное разрушение содержимого хотя бы одной используемой ячейки памяти может вывести машину из строя;
- существует четкое соответствие между оперируемыми абстрактными объектами (числами, словами, записями) и аппаратурным составом вычислительного комплекса, т.е. каждому такому объекту можно поставить в соответствие определенную ячейку памяти.

Искусственная нейронная сеть (ИНС) — это взаимосвязанная совокупность простых обрабатывающих элементов (нейронов или узлов), которые с определенной (практически обусловленной) степенью приближения реализуют

функции нейронов живых организмов. Нейронные сети обладают способностью к обучению, т. е. являются *информационно активными системами.*

Подобно биологической нейронной системе ИНС является вычислительной системой с огромным числом параллельно функционирующих простых процессоров с множеством связей. Модели ИНС в некоторой степени воспроизводят «организационные» принципы, свойственные мозгу человека.

Глубокое изучение ИНС, с одной стороны, требует знания нейрофизиологии, науки о познании, психологии, физики (статистической механики), теории управления, теории вычислений, проблем искусственного интеллекта, статистики, распознавания образов, компьютерного зрения, параллельных вычислений и аппаратных средств ВТ. С другой стороны, ИНС также стимулируют эти дисциплины, обеспечивая их новыми инструментами и представлениями. Этот симбиоз жизненно необходим для исследований по нейронным сетям.

Представим некоторые проблемы, решаемые в контексте ИНС и представляющие интерес для ученых и инженеров.

Классификация образов. Задача состоит в указании принадлежности входного образа (например, речевого сигнала или рукописного символа), представленного вектором признаков, одному или нескольким предварительно определенным классам. К известным приложениям относятся распознавание букв, распознавание речи, классификация сигнала электрокардиограммы, классификация клеток крови.

Кластеризация/категоризация. При решении задачи кластеризации, которая известна также как классификация образов «без учителя», отсутствует обучающая выборка с метками классов. Алгоритм кластеризации основан на подобии образов и размещает близкие образы в один кластер. Известны случаи применения кластеризации для извлечения знаний, сжатия данных и исследования свойств данных.

Аппроксимация функций. Предположим, что имеется обучающая выборка $((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$ (пары данных «вход-выход»), которая генерируется неизвестной функцией (x), искаженной шумом. Задача аппроксимации состоит в нахождении оценки неизвестной функции (x). Аппроксимация функций необходима при решении многочисленных инженерных и научных задач моделирования.

Предсказание/прогноз. Пусть заданы n дискретных отсчетов $\{y(t_1), y(t_2), ..., ..., y(t_n)\}$ в последовательные моменты времени $t_1, t_2, ..., t_n$. Задача состоит в предсказании значения $y(t_{n+1})$ в некоторый будущий момент времени t_{n+1}. Предсказание/прогноз имеют значительное влияние на принятие решений в бизнесе, науке и технике. Предсказание цен на фондовой бирже и прогноз погоды являются типичными приложениями техники предсказания/прогноза.

Оптимизация. Многочисленные проблемы в математике, статистике, технике, науке, медицине и экономике могут рассматриваться как проблемы оптимизации. Задачей алгоритма оптимизации является нахождение такого решения, которое удовлетворяет системе ограничений и максимизирует или минимизирует

целевую функцию. Задача коммивояжера, относящаяся к классу NP-полных, является классическим примером задачи оптимизации.

Память, адресуемая по содержанию. В модели вычислений фон Неймана обращение к памяти доступно только посредством адреса, который не зависит от содержания памяти. Более того, если допущена ошибка в вычислении адреса, то может быть найдена совершенно иная информация. Ассоциативная память, или память, адресуемая по содержанию, доступна по указанию заданного содержания. Содержимое памяти может быть вызвано даже по частичному входу или искаженному содержанию. Ассоциативная память чрезвычайно желательна при создании мультимедийных информационных баз данных.

Управление. Рассмотрим динамическую систему, заданную совокупностью $\{u(t), y(t)\}$, где $u(t)$ является входным управляющим воздействием, а $y(t)$ — выходом системы в момент времени t. В системах управления с эталонной моделью целью управления является расчет такого входного воздействия $u(t)$, при котором система следует по желаемой траектории, диктуемой эталонной моделью. Примером является оптимальное управление двигателем.

6.6.2. Нейронная сеть человека

Нейрон (нервная клетка) является особой биологической клеткой, которая обрабатывает информацию (рис. 6.5). Она состоит из **тела клетки (cell body)** или **сомы (soma)**, и двух типов внешних древоподобных ветвей: **аксона (axon)** и **дендритов (dendrites)**. Тело клетки включает **ядро (nucleus)**, которое содержит информацию о наследственных свойствах, и плазму, обладающую молекулярными средствами для производства необходимых нейрону материалов. Нейрон получает сигналы (импульсы) от других нейронов через дендриты (приемники) и передает сигналы, сгенерированные телом клетки, вдоль аксона (передатчик), который в конце разветвляется на **волокна (strands)**. На окончаниях этих волокон находятся **синапсы (synapses)**.

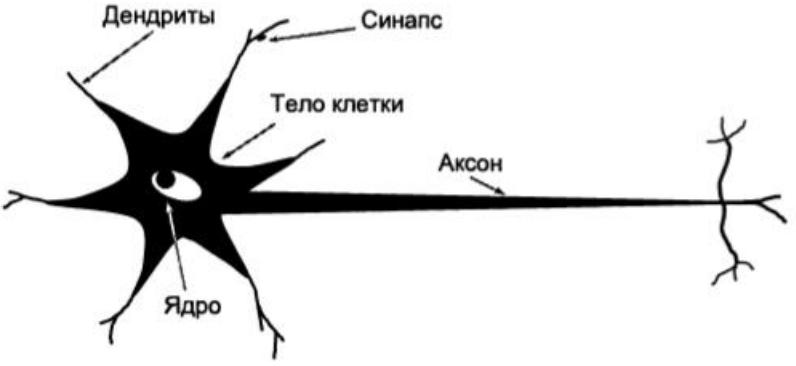


Рис. 6.5. Человеческий нейрон

Синапс является элементарной структурой и функциональным узлом между двумя нейронами (волокно аксона одного нейрона и дендрит другого). Когда импульс достигает синаптического окончания, высвобождаются определенные химические вещества, называемые **нейротрансмиттерами**. Нейротрансмиттеры диффундируют через синаптическую щель, возбуждая или затормаживая, в зависимости от типа синапса, способность нейрона-приемника генерировать электрические импульсы. Результативность синапса может настраиваться проходящими через него сигналами, так что синапсы могут обучаться в зависимости от активности процессов, в которых они участвуют. Эта зависимость от предыстории действует как память, которая, возможно, ответственна за память человека.

Кора головного мозга человека является протяженной, образованной нейронами поверхностью толщиной от 2 до 3 мм с площадью около 2200 $см^2$, что вдвое превышает площадь поверхности стандартной клавиатуры. Кора головного мозга содержит около 10^{11} нейронов, что приблизительно равно числу звезд Млечного пути. Каждый нейрон связан с 10^3–10^4 другими нейронами. В целом мозг человека содержит приблизительно от 10^{14} до 10^{15} взаимосвязей.

Нейроны взаимодействуют посредством короткой серии импульсов, как правило, продолжительностью несколько миллисекунд. Сообщение передается посредством частотно-импульсной модуляции. Частота может изменяться от нескольких единиц до сотен герц, что в миллион раз медленнее, чем самые быстродействующие переключательные электронные схемы. Тем не менее сложные решения по восприятию информации, как, например, распознавание лица, человек принимает за несколько сотен миллисекунд. Эти решения контролируются сетью нейронов, которые имеют время выполнения операций всего несколько миллисекунд. Это означает, что вычисления требуют не более 100 последовательных стадий. Другими словами, для таких сложных задач мозг «запускает» параллельные программы, содержащие около 100 шагов. Это известно, как «правило ста шагов». Рассуждая аналогичным образом, можно обнаружить, что количество информации, посылаемое от одного нейрона другому, должно быть очень маленьким (несколько бит). Отсюда следует, что основная информация не передается непосредственно, а захватывается и распределяется в связях между нейронами.

Основой отражения реальности является воспринятое ощущение в совокупности с информацией, полученной и сохраненной в нервной (нейронной) системе в результате предыдущего опыта (обучения).

Наконец, стоит отметить, что синапсы являются аналоговыми преобразователями информации, а наличие или отсутствие импульса в нервном волокне может интерпретироваться как цифровое кодирование. При этом важно учесть, что в нервных системах животных и людей кодируются и декодируются не только физические и химические свойства раздражителя, но и биологическое значение раздражителя для поддержания жизни организма, его развития и размножения.

6.6.3. Простейшие нейрокомпьютеры

В простейшем случае **нейрокомпьютер** представляет собой сеть **элементарных процессоров** (ЭП), называемых по биологической терминологии **нейронами**, объединенных в одно- или многослойную структуру. В идеальном случае все нейроны связаны между собой линиями передачи данных по принципу «каждый с каждым».

Нейрон, получив данные от других нейронов, с которыми он связан, выполняет вычисления некоторой функции F, значениями которой могут быть в простейшем случае 0 либо 1.

Связям, идущим от других нейронов к некоторому i-му нейрону, присваиваются весовые коэффициенты W_i. Эти весовые коэффициенты, умноженные на значения, поступающие от других нейронов, суммируются, и полученная сумма становится аргументом функции, вычисляемой принимающим нейроном.

В каждый момент времени все нейроны работают одновременно, обеспечивая максимальный параллелизм нейрокомпьютерной системы.

Схематически один i-й элементарный процессор (ЭП) нейросети, или нейрон, можно представить так, как это показано на рис. 6.6.

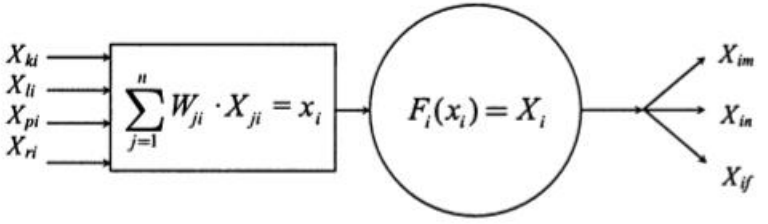


Рис. 6.6. Схема нейрона (процессорного элемента нейрокомпьютера): W_{ji} — веса, поступающие на i-й ЭП от j-го ЭП; X_{ij} — значение функции F_i, рассылаемое другим нейронам, связанным с данным

Веса назначаются при подготовке (настройке) нейросети к работе. Эта настройка происходит по некоторым алгоритмам «обучения» сети.

В качестве функции $F_i(x_i) = X_i$ чаще всего используют пороговую функцию, принимающую значения 0 или 1 и определяемую следующим образом:

$$F_i(x_i) = \begin{cases} 0, \text{ если } x_i \le V_i, \\ 1, \text{ если } x_i > V_i. \end{cases} \qquad (6.3)$$

Параметр V_i называется порогом. Пороговая функция переводит нейрон в два состояния — активное или пассивное. Функция $F_i(x_i) = X_i$ называется иначе **функцией активации**.

Нейрокомпьютерная сеть обычно работает по тактам (дискретно). За такт происходит вычисление одновременно всех функций $F_i(x_i) = X_i$ При этом новые значения этих функций будут участвовать в следующем такте работы нейронной

сети. Если функции $F_i(x_i) = X_i$ таковы, что могут принимать только два значения, например 0 или 1, то они называются **бинарными**. Нейрон, принявший значение 0, становится пассивным. Это равносильно тому, что он как бы перестает передавать информацию другим, связанным с ним нейронам. Во многих нейросетях полагают все функции F_i однотипными для всех нейронов.

В нейросетевых вычислениях значение выхода функции $F_i(x_i) = X_i$ не обязательно дискретно и не обязательно определяется порогом. Функция активации F_i может быть задана аналитически различным образом (например, в виде линейной функции или сигмоиды).

Считается, что сеть пришла в равновесное состояние, когда от шага к шагу ее работы перестают изменяться значения выходов. Это эквивалентно тому, что некий итерационный процесс сошелся. Распределяя нужным образом веса связей и варьируя правила вычисления функций всех узлов, можно настраивать такую сеть на решение широкого круга задач указанных выше классов.

Нейронные сети можно «обучать», автоматически изменяя веса и значения порогов таким образом, чтобы на обучающих примерах сеть давала правильный ответ на поставленный вопрос. В этом случае можно надеяться на то, что сеть будет давать правильные ответы на аналогичные вопросы, касающиеся других объектов с другими параметрами.

6.6.4. Архитектура нейронной сети

По аналогии с биологической нейронной сетью искусственную нейронную сеть можно представить как направленный граф со взвешенными связями, в котором искусственные нейроны являются узлами.

По архитектуре связей ИНС могут быть сгруппированы в два класса: **сети прямого распространения**, в которых графы не имеют петель, и **рекуррентные сети**, или сети с обратными связями.

В наиболее распространенном семействе сетей первого класса, называемых **многослойным персептроном**, нейроны расположены слоями и имеют однонаправленные связи между слоями. На рис. 6.7 представлены типовые сети каждого класса. Сети прямого распространения являются статическими в том смысле, что на заданный вход они вырабатывают одну совокупность выходных значений, не зависящих от предыдущего состояния сети. Рекуррентные сети являются динамическими, так как в силу обратных связей в них модифицируются входы нейронов, что приводит к изменению состояния сети.

6.6.4.1. Обучение

Способность к обучению является фундаментальным свойством мозга. В контексте ИНС процесс обучения может рассматриваться как настройка архитектуры сети и весов связей для эффективного выполнения специальной задачи. Обычно нейронная сеть должна настроить веса связей по имеющейся обучающей выборке. Функционирование сети улучшается по мере итеративной настройки весовых

коэффициентов. Свойство сетей обучаться на примерах делает их более привлекательными по сравнению с системами, которые следуют определенным правилам функционирования, сформулированным экспертами.

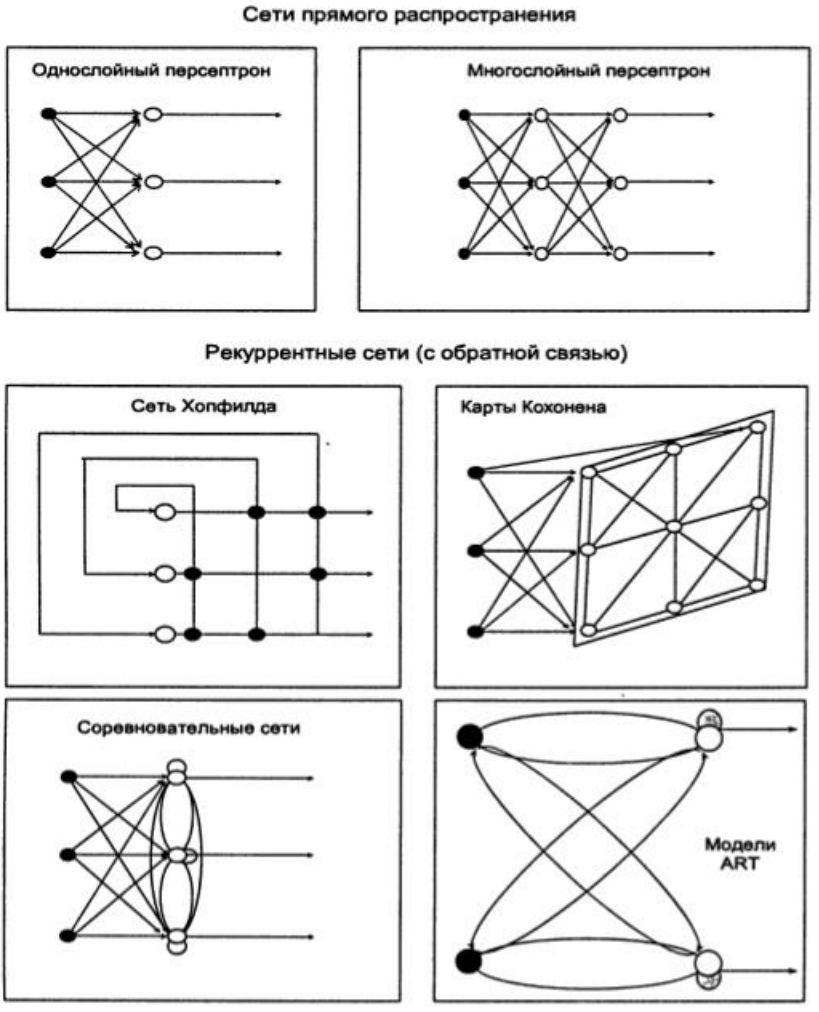


Рис. 6.7. Систематизация архитектур сетей прямого распространения и рекуррентных (с обратной связью)

Для конструирования процесса обучения, во-первых, необходимо иметь модель внешней среды, в которой функционирует нейронная сеть, — знать доступную для сети информацию. Эта модель определяет **парадигму обучения**. Во-вторых, необходимо понять, как модифицировать весовые параметры сети — какие *правила обучения* управляют процессом настройки. Алгоритм обучения означает процедуру, в которой используются правила обучения для настройки весов.

Существуют три парадигмы обучения: **«с учителем»**, **«без учителя»** (самообучение) и **смешанная**. В первом случае нейронная сеть располагает правильными ответами (выходами сети) на каждый входной пример. Веса настраиваются так, чтобы сеть производила ответы как можно более близкие к известным правильным ответам. Усиленный вариант обучения с учителем предполагает, что известна только критическая оценка правильности выхода нейронной сети, но не сами правильные значения выхода.

Обучение без учителя не требует знания правильных ответов на каждый пример обучающей выборки. В этом случае раскрывается внутренняя структура данных или корреляции между образцами в системе данных, что позволяет распределить образцы по категориям. При смешанном обучении часть весов определяется посредством обучения с учителем, в то время как остальная получается с помощью самообучения.

Теория обучения рассматривает три фундаментальных свойства, связанных с обучением по примерам: *емкость*, *сложность образцов* и *вычислительная сложность*. Под **емкостью** понимается, сколько образцов может запомнить сеть, и какие функции и границы принятия решений могут быть на ней сформированы. **Сложность образцов** определяет число обучающих примеров, необходимых для достижения способности сети к обобщению. Слишком малое число примеров может вызвать «переобученность» сети, когда она хорошо функционирует на примерах обучающей выборки, но плохо — на тестовых примерах, подчиненных тому же статистическому распределению. Известны 4 основных типа правил обучения: *коррекция по ошибке*, *машина Больцмана*, *правило Хебба* и обучение *методом соревнования*.

Правило коррекции по ошибке. При обучении с учителем для каждого входного примера задан желаемый выход d. Реальный выход сети y может не совпадать с желаемым. Принцип коррекции по ошибке при обучении состоит в использовании сигнала $(d - y)$ для модификации весов, обеспечивающей постепенное уменьшение ошибки. Обучение имеет место только в случае, когда перцептрон ошибается.

Обучение Больцмана. Представляет собой стохастическое правило обучения, которое следует из информационных теоретических и термодинамических принципов. Целью обучения Больцмана является такая настройка весовых коэффициентов, при которой состояния видимых нейронов удовлетворяют желаемому распределению вероятностей. Обучение Больцмана может рассматриваться как специальный случай коррекции по ошибке, в котором под ошибкой понимается расхождение корреляций состояний в двух режимах.

Правило Хебба. Самым старым обучающим правилом является постулат обучения Хебба. Хебб опирался на следующие нейрофизиологические наблюдения: если нейроны с обеих сторон синапса активизируются одновременно и регулярно, то сила синаптической связи возрастает. Важной особенностью этого правила яв-

ляется то, что изменение синаптического веса зависит только от активности нейронов, которые связаны данным синапсом.

Обучение методом соревнования. В отличие от обучения Хебба, в котором множество выходных нейронов могут возбуждаться одновременно, при соревновательном обучении выходные нейроны соревнуются между собой за активизацию. Это явление известно, как правило «победитель берет все». Подобное обучение имеет место в биологических нейронных сетях. Обучение посредством соревнования позволяет кластеризовать входные данные: подобные примеры группируются сетью в соответствии с корреляциями и представляются одним элементом.

Можно заметить, что сеть может обучаться бесконечно. Некоторый входной образец может активизировать другой выходной нейрон на последующих итерациях в процессе обучения. Это ставит вопрос об *устойчивости обучающей системы*. Система считается **устойчивой**, если ни один из примеров обучающей выборки не изменяет своей принадлежности к категории после конечного числа итераций обучающего процесса. Один из способов достижения стабильности состоит в постепенном уменьшении скорости обучения. Однако это искусственное торможение обучения вызывает другую проблему, называемую **пластичностью** и связанную со способностью адаптации к новым данным. Особенности обучения методом соревнования известны под названием дилеммы стабильности-пластичности Гроссберга.

6.6.4.2. Модели сетей

Теоретически основы нейроматематики были заложены уже в начале 40-х годов. В 1943 г. У. МакКаллок и его ученик У. Питтс сформулировали основные положения теории деятельности головного мозга. В результате их исследований была разработана модель нейрона как простейшего процессорного элемента, выполняющего вычисление переходной функции от скалярного произведения вектора входных сигналов и вектора весовых коэффициентов, и предложена конструкция сети таких элементов для выполнения логических и арифметических операций. Ими же было сделано основополагающее предположение о том, что такая сеть способна обучаться, распознавать образы, обобщать полученную информацию.

Несмотря на то что за прошедшие годы нейроматематика ушла далеко вперед, многие утверждения МакКаллока остаются актуальными и поныне. В частности, при большом разнообразии моделей нейронов принцип их действия, заложенный МакКаллоком и Питтсом, остается неизменным.

Недостатком данной модели является сама модель нейрона — «пороговый» вид переходной функции. В формализме У. МакКаллока и У. Питтса нейроны имеют состояние 0 или 1 и пороговую логику перехода из состояния в состояние. Каждый нейрон в сети определяет взвешенную сумму состоянии всех других нейронов и сравнивает ее с порогом, чтобы определить свое собственное состояние. Пороговый вид функции не предоставляет нейронной сети достаточную гибкость при обучении и настройке на заданную задачу. Если значение вычисленного ска-

лярного произведения, даже незначительно не достигает до заданного порога, то выходной сигнал не формируется вовсе и нейрон «не срабатывает». Это значит, что теряется интенсивность выходного сигнала данного нейрона и, следовательно, формируется невысокое значение уровня на взвешенных выходах в следующем слое нейронов.

Серьезное развитие нейрокибернетика получила в работах американского нейрофизиолога Френсиса Розенблатта (Корнельский университет). В 1958 г. он предложил свою модель нейронной сети. Розенблатт ввел в модель МакКаллока и Питтса способность связей к модификации, что сделало ее обучаемой. Эта модель была названа **персептроном**. Первоначально персептрон представлял собой однослойную структуру с жесткой пороговой функцией процессорного элемента и бинарными или многозначными входами. Первые персептроны были способны распознавать некоторые буквы латинского алфавита. Впоследствии модель персептрона была значительно усовершенствована.

Персептрон применялся для задачи автоматической классификации, которая в общем случае состоит в разделении пространства признаков между заданным количеством классов. В двухмерном случае требуется провести линию на плоскости, отделяющую одну область от другой. Персептрон способен делить пространство только прямыми (плоскостями).

Серьезным недостатком персептрона является то, что не всегда существует такая комбинация весовых коэффициентов, при которой имеющееся множество образов будет распознаваться данным персептроном. Причина этого недостатка состоит в том, что лишь небольшое количество задач предполагает, что линия, разделяющая эталоны, будет прямой. Обычно это достаточно сложная кривая, замкнутая или разомкнутая. Если учесть, что однослойный персептрон реализует только линейную разделяющую поверхность, применение его там, где требуется нелинейная, приводит к неверному распознаванию (эта проблема называется линейной неразделимостью пространства признаков). Выходом из этого положения является использование многослойного персептрона, способного строить ломаную границу между распознаваемыми образами.

Описанная проблема не является единственной трудностью, возникающей при работе с персептронами, — также слабо формализован метод обучения персептрона.

В 70-е годы интерес к нейронным сетям значительно упал, однако работы по их исследованию продолжались. Был предложен ряд интересных разработок, таких, например, как **когнитрон**, способный хорошо распознавать достаточно сложные образы (иероглифы и т. п.) независимо от поворота и изменения масштаба изображения. Автором когнитрона является японский ученый И. Фукушима.

Новый виток быстрого развития моделей нейронных сетей, который начался с 1990-х годов, связан с работами Амари, Андерсона, Карпентера, Кохонена и других, и в особенности, Хопфилда, а также под влиянием обещающих успехов

оптических технологий и зрелой фазы развития СБИС для реализации новых архитектур.

Кохонен, например, предложил использовать самоорганизующиеся карты, обладающие свойством сохранения топологии, которое воспроизводит важный аспект карт признаков в коре головного мозга человека. По существу, сеть Кохонена представляет собой двумерный массив элементов, когда каждый элемент связан со всеми входными узлами. Эта сеть является частным случаем сети, обучающейся методом соревнования. Как правило, самоорганизующиеся карты Кохонена могут быть использованы при проектировании многомерных данных, аппроксимации плотности и кластеризации. Известны случаи, когда сеть Кохонена успешно использовалась для распознавания речи, обработки изображений, в робототехнике, задачах управления.

Начало современному математическому моделированию нейронных вычислений было положено работами Хопфилда в 1982 г., в которых была математически сформулирована модель ассоциативной памяти на нейронной сети. Но не столько сама модель послужила толчком к появлению работ других авторов на эту тему, сколько введенная Хопфилдом **функция вычислительной энергии** нейронной сети. Это аналог функции Ляпунова в динамических системах. Показано, что для однослойной нейронной сети со связями типа «все на всех» характерна сходимость к одной из конечного множества равновесных точек, которые являются локальными минимумами функции энергии, содержащей в себе всю структуру взаимодействий в сети. Понимание такой динамики в нейронной сети было и у других исследователей. Однако Хопфилд и Тэнк показали, как конструировать функцию энергии для конкретной оптимизационной задачи и как использовать ее для отображения задачи в нейронную сеть. Этот подход получил развитие и для решения других комбинаторных оптимизационных задач. Привлекательность подхода Хопфилда состоит в том, что нейронная сеть для конкретной задачи может быть запрограммирована без обучающих итераций.

Развитием модели Хопфилда для решения комбинаторных оптимизационных задач и задач искусственного интеллекта является машина Больцмана, предложенная и исследованная Джефери Е. Хинтоном и Р. Земелом. В ней, как и в других моделях, нейрон имеет состояния 1 или 0, и связь между нейронами обладает весом. Каждое состояние сети характеризуется определенным значением функции консенсуса (аналог функции энергии). Максимум функции консенсуса соответствует оптимальному решению задачи.

6.6.5. Промышленные нейрокомпьютеры и перспективы их развития

Первый нейрокомпьютер был создан в конце 50-х годов Ф. Розенблаттом (Корнельский университет г. Итаки, штат Нью-Йорк). Этот компьютер, получивший название персептрон, использовался для распознавания букв независимо от их по-

ложения. ЭВМ Розенблатта была воплощением идеи У. МакКаллока и У. Питтса, у которых нейроны рассматривались как логические устройства.

В 1988 г. с участием фирмы Adaptive Solutions был разработан нейронный компьютер **CNAPS**. Этот компьютер был создан по SIMD-архитектуре, его сервер содержал 256 обрабатывающих процессоров. Каждый процессор имел свое ЗУ емкостью 4 Кбайт. Производительность компьютера CNAPS достигала 5,12 млрд коммутаций в секунду. В режиме обучения сервер работал с производительностью 1 млрд коммутаций в секунду. Каждый алгоритм усваивался компьютером CNAPS за 6 секунд. До конца 1991 г. фирмой было поставлено на рынок около 100 единиц нейрокомпьютера CNAPS.

Компания SNI (Siemens Nixdorf Informations), дочернее предприятие компании Siemens, в сотрудничестве с Маннгеймским университетом в 1994 г. создали нейрокомпьютер под названием **SYNAPSE 1**. В дальнейшем на рынок поступили модели **SYNAPSE 2** и **3**. Сфера применения этих нейрокомпьютеров: распознавание речи, изображений, образов, ускорение работы программных эмуляторов. Обучение нейрокомпьютера занимало около одного часа. Нейрокомпьютер представлял собой многопроцессорную систему с наращиваемой памятью.

В состав **SYNAPSE 2** входили:

- один нейрочип MA16 (40 Гц);
- сигнальный процессор TMS320C50 (55 МГц);
- модуль целочисленной обработки на базе TMS320C50 (55 МГц);
- память образов (Y-memore);
- память весов (W-memore).

В нейропроцессоре **SYNAPSE 3** имелись два процессора M16, типовая производительность одной нейроплаты **SYNAPSE 3** составляла 2,4 млрд оп/с. В качестве базовых ЭВМ использовались рабочие станции фирмы Sun. Габаритные размеры нейрокомпьютера составляли 667 × 398 × 680 мм.

На современном рынке изделия, основанные на использовании механизма действия нейронных сетей, представлены в виде нейроплат. В качестве типичного примера нейроплаты можно назвать плату **MB S6232** японской фирмы Fujitsu. На плате размещены процессор цифровой обработки сигналов и оперативная память емкостью 4 Мбайт, что позволяет использовать такую плату для реализации ИНС, содержащей до тысячи нейронов.

Большинство современных нейрокомпьютеров представляют собой просто персональный компьютер или рабочую станцию, в состав которых входит дополнительная нейроплата. К их числу относятся, например, компьютеры серии **FMR** фирмы Fujitsu. Возможностей подобных систем достаточно для разработки новых алгоритмов и решения большого числа прикладных задач методами нейроматематики.

Отметим отечественные достижения в области построения нейрокомпьютеров. В НТЦ «Модуль» за период с 1989 по 1999 г. были разработаны многопроцес-

сорные ускорительные платы **МЦ5.001** и **МЦ5.002**. Последняя содержит до 6 процессоров TMS320C40, до 20 Мбайт статической памяти на каждый процессор и 64 Мбайт динамической ОП. Общая производительность — до 300 MFLOPS. Плата выполнена в конструктиве VME, что позволяет использовать ее в бортовых системах, расположенных на летательном аппарате.

Несмотря на определенные достижения в области создания нейрокомпьютеров, широкого распространения они не получили. На сегодняшний день экономически выгоднее реализовывать нейроалгоритмы программно на универсальных ЭВМ. Существует достаточное количество программных пакетов (например, **Neural Bench**), с помощью которых можно реализовать ИНС под тот или иной алгоритм. Программные ИНС широко используются в системах распознавания текстов (OCR-системах).

Вопросы для самоконтроля

1. Что понимается под термином «нанотехнология»?
2. При каких размерах объектов не действуют законы классической физики?
3. Кем и когда был изобретен сканирующий туннельный микроскоп?
4. Что такое «нанотрубка»?
5. Сколько и какие периоды можно выделить в развитии биокристаллов?
6. Перечислите основные структурные элементы молекулярного компьютера.
7. Что понимается под термином «оптический компьютер»?
8. Перечислите основные элементы гибридной ВС.
9. Что понимается под аббревиатурами: SEED и S-SEED?
10. По каким направлениям в настоящее время ведутся работы по созданию оптической ЭВМ?
11. Что такое квантовый компьютер?
12. Что понимается под термином «кубит»?
13. Изобразите структурную схему квантового компьютера.
14. Перечислите основные требования, предъявляемые к физической среде КК.
15. Что такое криогенная ЭВМ?
16. Что собой представляет нейронная сеть человека?
17. Какие структуры искусственных нейронных сетей используют в настоящее время?
18. Что такое «функция активации»?
19. Как может обучаться нейросеть?
20. Перечислите основные модели нейросетей.
21. Каковы основные достижения и перспективы развития нейрокопьютеров?

Литература

1. ANSIx3.253-2002: Information Technology-SCSI-3 Parallel Interface (SPI), X3T10/855D. New York: American National Standarts Institute, 2002.

2. *Deutsch D.* Quantum computational networks. — London: Proc. R. Soc., 1989.

3. *Deutsch D.* Quantum theory, the Church-Turing principle and the universal quantum computer. — London: Proc. R. Soc., 1985.

4. *Feynman R.* Quantum mechanical computers // Optic News. — 1985. — February. — № 11.

5. *Hennessy J. L., Patterson D. A.* Computer architecture: A Quantitative Approach. 2nd Edition. — San Francisco: Morgan Kaufmann Publishers, 1996.

6. Serial attached SCSI // http://www.fcenter.ru/online.shtml?articles/hardware/hdd/11080.

7. *Shor P. W.* Algorithms for Quamtum Computation: Discrete log and Factoring // Proceedings of the 35th Annual Symposium on the Foundations of Computer Science, edited by *S. Goldwasser.* — Los Alamitos: IEEE Computer Society Press. — 1994.

8. *Sterling T., Becker D., Savarese D., et al.* Beowulf: A Parallel Workstation for Scientific Computation // Proceeding of the 1995 International Conference on Parallel Processing (ICPP). — 1995. — August. — Vol. 1.

9. *Yao A. C.-C.* Quantum circuit complexity // Proceedings of the 34th Annual Symhosium on the Foundations of Computer Science. — Los Alamitos: IEEE Computer Society Press. — 1993.

10. *Акулов Л. В., Борзенко Е. И., Новотельнов В. Н. и др.* Теплофизические свойства криопродуктов: учебное пособие для вузов. — СПб.: Политехника, 2001.

11. *Амамия М., Танака Ю.* Архитектура ЭВМ и искусственный интеллект / Пер. с яп. *Махарадзе С. О.* / Под ред. *Волкова Н. Г.* — М.: Мир, 1993.

12. *Андрианов А. Н., Ефимкин К. Н., Задыхайло И. Б.* Язык Норма / Препринт ИПМ им. М. В. Келдыша АН СССР. — 1985. — № 165.

13. Архитектура портативных компьютеров // Upgrade: новый уровень ваших компьютеров. — М: СК Пресс. — 2004. — № 3 (18).

14. *Барановский В.* Raid массивы начального уровня // http://www.citforum.ru/hardware/data/raid.

15. *Барский А. Б.* Параллельные процессы в вычислительных системах. Планирование и организация. — М.: Радио и связь, — 1990.

16. *Берман Г. П., Дулен Г. Д., Маньери Р., Цифринович В. И.* Введение в квантовые компьютеры / Пер. с англ. *Порсева В. Е.* / Под ред. *Кокина А. А.* — М.: Институт компьютерных исследований, 2004.

17. *Борзенко А.* Технология Super DLT // PCWeek/RE. — 2000. — № 45 (267).

18. *Бурцев В. С.* Новые принципы организации вычислительных процессов высокого параллелизма // Труды Первой Всероссийской научной конференции «Методы и средства обработки информации». — М.: Изд-во МГУ. — 2003.

19. В свете лазерного луча // Upgrade: новый уровень ваших компьютеров. — М: СК Пресс. — 2005. — № 5 (24).

20. *Валиев К. А.* Квантовая информатика: компьютеры, связь и криптография // Вестник РАН. — 2000. — Т. 70. — № 8.

21. *Валиев К. А.* Квантовые компьютеры // Открытые системы. — 2000. — № 5–6.

22. *Валиев К. А., Кокин А. А.* Из итогов XX века: от кванта к квантовым компьютерам // Природа. — 2002. — № 12.

23. *Васильковский В. А., Котов В. Е., Марчук А. Г., Миренков Н. Н.* Автоматизация параллельного программирования. — М.: Радио и связь, 1983.

24. Вводим изображение // Upgrade: новый уровень ваших компьютеров. — М: СК Пресс. — 2005. — № 5 (24).

25. *Виксне П., Фомин Д., Черников В.* Однокристальный цифровой нейропроцессор с переменной разрядностью операндов // Приборостроение. — Т. 36. — 1996.— № 7.

26. *Власов А. И.* Аппаратная реализация нейровычислительных управляющих систем // Приборы и системы управления. — 1999. — № 2.

27. *Власов А. И.* Нейросетевая реализация микропроцессорных систем активной акусто- и виброзащиты // Нейрокомпьютеры: разработка и применение. — 2000. — № 1.

28. *Воеводин В. В., Воеводин Вл. В.* Параллельные вычисления. — СПб.: БХВ-Петербург, 2002.

29. *Воеводин Вл. В.* Параллельная обработка данных. Курс лекций // http://www.parallel.ru/info/education/vvv_course.html.

30. *Воеводин Вл. В., Капитонова А. П.* Методы описания и классификации вычислительных систем. — М.: Изд-во МГУ, 1994.

31. *Волков А. А., Угляренко В. П.* Управление распределением вычислительной нагрузки в сетях ЭВМ // Механизация и автоматизация управления. — 1982. — № 3.

32. Вычислительные машины, системы и сети / Под. ред. *Пятибратова А. П.* — М.: Финансы и статистика, 1991.

33. *Гаврилкевич М. В.* Введение в нейроматематику // Обозрение прикладной и промышленной математики. — М.: ТВП, 1994.

34. *Галушкин А. И.* Некоторые исторические аспекты развития элементной базы вычислительных систем с массовым параллелизмом (80-е и 90-е годы) // Нейрокомпьютер. — 2000. — № 1.

35. *Горбань А. Н., Россиев Д. А.* Нейронные сети на персональном компьютере. — Новосибирск: Наука. Сибирская издательская фирма РАН, 1996.

36. Графические адаптеры: четвертое поколение // Upgrade: новый уровень ваших компьютеров. — М: СК Пресс. — 2005. — № 2 (21).

37. Гузик В. Ф., Каляев В. А., Костюк А. И. Организация ЭВМ и систем: Учебное пособие. — Таганрог: ТРТУ, 1999.

38. Гук М. Аппаратные средства IBM PC: Энциклопедия. — Спб.: Питер, 2006.

39. Гуров В. В., Чуканов В. О. Архитектура и организация ЭВМ. Курс лекций // http://www.intuit.ru/department/hardware/archhard2.

40. Гусев А. И. Наноматериалы, наноструктуры, нанотехнологии. — М.: Физматлит, 2005.

41. Дагаев А. А. Союз сильных: новые тенденции международного технологического развития // Рос. предпринимательство. — 2003. — № 4.

42. Денисов О., Нивников Д. Мультимедийные ПК // Компьютер сегодня. — 2006. — № 2.

43. Евреинов Э. В., Косарев Ю. Г. Однородные универсальные вычислительные системы высокой производительности. — Новосибирск: Наука,1966.

44. Егоров А. RAID-массив и резервное копирование // http://www.timcompany.ru/article5.html.

45. Забродин А. В., Левин В. К., Сидоров А. Ф., Лацис А. О. и др. Семейство многопроцессорных вычислительных систем МВС-100. — М.: ИПМ им. М. В. Келдыша РАН, НИИ «Квант», ИММ УрО РАН, 1995.

46. Исихара С. Оптические компьютеры. Новый век науки / Пер. с англ. *Богдасарова С. В.* / Под ред. *Воронцова М. А.* — М.: Наука, 1992.

47. Каган Б. М. Электронные вычислительные машины и системы: Учебное пособие для вузов. 2-е изд., перераб. и доп. — М.: Энергоатомиздат, 1985.

48. Кирсанов Э. Ю. Организация ЭВМ и систем. Учебное пособие. — Казань: ТИСБИ, 2002.

49. Кирсанов Э. Ю. Цифровые нейрокомпьютеры: Архитектура и схемотехника / Под ред. *Галушкина А. И.* — Казань: КГУ. 1995.

50. Колисниченко Д. Оптические процессоры // http://dkws.narod.ru/linux/etc/optical/cpu.html.

51. Колисниченко О. В., Шишигин И. В. Аппаратные средства PC. 5-е изд., перераб. и доп. — Спб.: БХВ-Петербург. 2004.

52. Комарцева Л. Г., Максимов А. В. Нейрокомпьютеры. — М.: Изд-во МГТУ им. Н. Э. Баумана. — 2004.

53. Коновалов Н. А., Крюков В. А., Михайлов С. Н., Погребцов А. А. Fortran DVM — язык разработки мобильных параллельных программ // Программирование. — 1995.— № 1.

54. Копейкин М. В., Спиридонов В. В., Шумова Е. О. Организация ЭВМ и систем (память ЭВМ): Учебное пособие. — Спб.: СЗТУ, 2004.

55. Корнеев В. В. Вычислительные системы. — М: Гелиос АРВ, 2004.

56. Корнеев В. В. Параллельные вычислительные системы. — М: Нолидж, 1999.

57. Ларионов А. М., Майоров С. А., Новиков Г. И. Вычислительные комплексы, системы и сети. — Л.: Энергоатомиздат,1987.

58. Лацис А. О. Как построить и использовать суперкомпьютер. — М.: Бестселлер, 2003.

59. Липаев В. В. Распределение ресурсов в вычислительных системах. — М.: Статистика, 1979.

60. Льюис Т. Мэйнфрейм умер. Да здравствует мэйнфрейм! // Открытые системы. — 1999. — № 9–10.

61. Малых Н. Интерфейс IDE // Электронная библиотека компании BiLiM Ltd. — http://www.citforum.ru/hardware/bookide/index.shtml.

62. Манин Ю. И. Вычислимое и невычислимое. — М.: Советское радио, 1980.

63. Миренков Н. Н. Параллельное программирование для многомодульных вычислительных систем. — М.: Радио и связь, 1989.

64. Миренков Н. Н. Управление памятью и процессорами в однородной вычислительной системе // Программирование. — 1976. — № 1.

65. Михайлов В. И., Князев Г. И., Макарычев П. П. Запоминающие устройства на оптических дисках. — М.: Радио и связь, 1991.

66. Могилев А. В. и др. Информатика: Учебное пособие. — М.: Академия, 2000.

67. Неволин В. К. Зондовые технологии в электронике. — М.: Техносфера, 2005.

68. Неизвестный И. Г. Квантовый компьютер и его полупроводниковая элементарная база // http://psj.nsu.ru/lector/neizvestniy.

69. Никитич А. Разработка устройства для ручного ввода символов в ЭВМ // http://neo-era.net/kb.a.xml.

70. Основы вычислительных систем: Учебное пособие // http://256bit.ru/education/infor1/index.htm.

71. Патий Е. Шина PCI Express: утопия или общая стандартизация? // Экспресс-электроника. — 2005. — № 1–2.

72. Поздняков Л. А., Храмцов М. Ю. Мобильная система программирования Фортран GNS для многопроцессорных систем с распределенной памятью // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. — 1996. — Вып. 4.

73. Проблемы построения и обучения нейронных сетей / Под ред. *Галушкина А. И.* и *Шахнова В. А.* — М.: Машиностроение, 1999.

74. Пул Ч., Оуэнс Ф. Нанотехнологии. — М.: Техносфера, 2004.

75. Пьянзин К. Состояние рынка аппаратных средств резервного копирования и архивирования // LAN. — 2000. — № 4.

76. Радужная капель // Upgrade: новый уровень ваших компьютеров. — М: СК Пресс. — 2005. — № 5 (24).

77. Рожденные обслуживать // Upgrade: новый уровень ваших компьютеров. — М: СК Пресс. — 2004. — № 1 (16).

78. Рыбалкина М. Нанотехнологии для всех. — Nanotechnology News Network, 2005.

79. Санько С. Квантовые вычисления: зачем это нужно? // Quanta et Qualia. — 2002. — № 39.

80. Суздалев И. П. Нанотехнология: физико-химия нанокластеров, наноструктур и наноматериалов. — М.: Эдиториал УРСС, 2006.

81. Таненбаум Э. Архитектура компьютера. 4-е изд. — М.: Питер, 2002.

82. Таненбаум Э., ван Стеен М. Распределенные системы: принципы и парадигмы. — М.: Питер, 2003.

83. Технологии флэш-памяти // http://www.ixbt.com/storage/flash-tech.shtml.

84. Типы современных ЖК-мониторов // Upgrade: новый уровень ваших компьютеров. — М: СК Пресс. — 2004. — № 2 (17).

85. Тихонов В. А. Краткий очерк развития СВТ. Монография. — М: Изд-во в/ч № 33965, 2000.

86. Тихонов В. А. Организация ЭВМ и систем: вводная лекция. — М.: Изд-во в/ч № 33965, 2005.

87. Тихонов В. А., Заикин В. В. Перспективные направления в развитии ЭВМ и вычислительных систем: обзорные лекции. — М.: Изд-во в/ч № 33965.

88. Тихонов В. А., Рудаков М. В. Оптические ЭВМ. Состояние и перспективы развития: обзорная лекция. — М.: Изд-во в/ч № 33965, 1999.

89. Тихонов В. А. Молекулярные ЭВМ: обзорная лекция. — М.: Изд-во в/ч № 33965, 1999.

90. Трахтенгерц Э. А. Введение в теорию анализа и распараллеливания программ ЭВМ в процессе трансляции. — М.: Наука, 1981.

91. Угрюмов Е. П. Цифровая схемотехника. — Спб.: БХВ, 2000.

92. Уинн Л. Рош. Библия по модернизации персонального компьютера. — Минск: Мир науки, 2003.

93. Федичкин Л. Квантовые компьютеры // Наука и жизнь. — 2001. — № 1.

94. Федотов В. Обзор flash-памяти на технологии Intel StrataFlash. Ч. 1 // http://www.ixbt.com/storage/flash-theory-p1.shtml.

95. Фейнман Р. Моделирование физики на компьютерах // Квантовый компьютер и квантовые вычисления. Т. 1. — Ижевск. — 1999. — № 2.

96. Фортран 90. Международный стандарт / Пер. с англ. *Дробышевич С. Г.* — М.: Финансы и статистика, 1998.

97. Французов Д. Оценка производительности вычислительных систем // Открытые системы. — 1996. — № 6.

98. Хамахер К., Вранешич З., Заки С. Организация ЭВМ. — СПб.: Питер, 2003.

99. Хехт-Нильсен Р. Нейрокомпьютинг: история, состояние, перспективы // Открытые системы. — 1998. — № 4.

100. Хоар Ч. Взаимодействующие последовательные процессы. — М.: Мир, 1989.

101. Цилькер Б. С., Орлов С. А. Организация ЭВМ и систем: учебник для вузов. — СПб.: Питер, 2004.

102. Цифровой звук: реализация // Upgrade: новый уровень ваших компьютеров. — М.: СК Пресс. — 2005. — № 1 (20).

103. Цифровой звук: теория // Upgrade: новый уровень ваших компьютеров. — М.: СК Пресс. — 2005. — № 1 (20).

104. Чеботарев А. USB: вчера, сегодня и завтра // http://www.citforum.ru/hardware/articles/usb/.

105. Чеканов Д. Реализация стандарта Serial ATA // http://www.3dnews.ru/storage/serial-ata.

106. Черняк Л. Шины от ISA до PCI Express // Computerworld. — 2005. — № 30.

107. Шарф С. В. Планирование прохождения задач на МВС-100 // Шестая конференция «Транскомпьютерные системы и их применение». Тезисы докладов. — Домодедово. — 1996.

108. Щукин Д. Оптические компьютеры // Новые технологии. — 2001. — № 5.

109. Энциклопедия flash-памяти // http://www.ak-cent.ru/?parent_id=9841.

110. Ястребова Е. В. Параллельные алгоритмы и транспьютеры: Учебно-методическое пособие. — М.: УРСС,1997.

111. Яценков В. С. Азбука CD и DVD: стандарты оптических дисков. — М.: Майор, 2004.

112. Ященко А. История развития IDE вплоть до ATA100 // http://www.ixbt.com/storage/ide-till-ata100.html.

AGP 152
CD-ROM 153, 155, 172, 173
dataflow 232, 233, 235, 312
FIFO 125, 148
LIFO 67
LRU 125, 147
MIDI 217
PCI 144
PIO 139
RAID 161
RAIT 180
RAMDAC 210
reduction 232, 233
TMDS 215
VGA 208, 209, 215
wavefront 232, 233
Автозагрузчик 179
адрес
— виртуальный 130
— исполнительный 60
— физический 130
адресация 60
— косвенная 80
— непосредственная 80
— по содержимому 96
— прямая 80
— регистровая 81
адресное пространство 97
— виртуальное 127
— реальное 128
— системы ввода-вывода 135
арбитраж 144
архитектура
— ARM 18
— CISC 63, 65
— MPP 234, 238
— NUMA 20, 234
— RISC 63, 65, 237
— SMP 20, 234, 238, 257
— VLIW 63
— ВС 10
— гарвардская 9
— классическая 57
— кластерная 238, 301
— КПК 18
— набора команд 8
— памяти ЭВМ 95
— параллельная 232
— потоковая 312
— принстонская 9
— системы 10
— систолическая 232, 317
— традиционная 57
— фон Неймана 57
— ЭВМ 8
Банк памяти 110
безотказность ЭВМ
или ВС 41
библиотека
— коммуникационная 254
— ленточная 179
биокристалл 327
бисекционная полнота 272
блок
— кэш-памяти 99
— памяти 98
буфер
— z-буфер 213
— преобразования адресов 129
БЭСМ-6 239
Вектор
— прерывания 91
— состояния процессора 86
— структурной живучести графа 281
векторизация программы 242
векторная обработка 220
ветвь параллельной программы 221
видеоадаптер 208

время
— восстановления 46
— дополнительное 96
— доступа 158
— доступа к памяти 96, 98
— запуска обмена 271
— наработки на отказ 45
— обслуживания прерывания 87
— ожидания 158
— ответа 40, 133
— отклика 203
— передачи данных 96, 158
— послесвечения 203
— реакции системы прерывания 87
— безотказной работы 161
— цикла обращения 100
Вычислительная сеть 9
вычислительная система 7
— ассоциативная 235, 266
— векторно-конвейерная 235, 260
— гибридная оптоэлектронная 333
— кластерная 301
— массово-параллельная 235, 238, 296
— матричная 235, 263
— многопроцессорная 7
— потоковая 309
— сильносвязанная 270
вычислительный комплекс (ВК) 7
Гамма-коррекция 205
глубина прерывания 87
головка 156
готовность ЭВМ или ВС 40, 46
граф
— зависимости 285
— межмодульных связей 280, 284
— потоков данных 310
Дедлок 284
дейзи-цепочка 89
джойстик 186
диаметр графа 281
дигитайзер 186
дискретизация 216
дисплей 200
дорожка 173
доступ
— ассоциативный 96, 116
— последовательный 95
— произвольный 95, 101
— прямой 95
— прямой к памяти 139
доступность ЭВМ или ВС 41
Жесткий диск 156
Зависимость по данным 244
задержка 95, 271
закон Амдала 225
запись
— обратная 290
— сквозная 290
запрос прерывания 84, 138
зерно параллелизма 220
Иерархия памяти 97
интенсивность восстановления 46
интенсивность отказов 44
интерфейс 9
— большой 133, 150
— малый 133, 153
искусственная нейронная сеть 348
Канал
— ввода-вывода 141
— мультиплексный 141
— неразделенный 141
— разделенный 141
— селекторный 141
картридж 180
каскад коммутатора 274
квантование 216
квантовые вычисления 339
клавиатура 182
классификация
— вычислительных систем 24
— Дункана 231
— карманных ПК 13

— команд 63
— мониторов 207
— накопителей на оптических дисках 170
— ноутбуков 16
— отказов 42
— параллельных ВС 233
— серверов 19
— сканеров 187
— Фенга 229
— Флинна 227
— ЭВМ 10
когерентность памяти 124, 289
код
— операции 60, 63
— прерывания 92
— Хэмминга 164
команды векторные 237
коммутатор
— баньян-сети 278
— Клоза 277
— простой 272
— распределенный составной 280
— составной 274
компьютер
— квантовый 339
— молекулярный 327
— оптический 332
— персональный 11
— суперкомпьютер 23
конвейерная обработка 258
контроллер
— ввода-вывода 136
— прямого доступа к памяти 139
контрольная сумма 167
коэффициент
— готовности 46
— попаданий 98
— распараллеливания 224
кубит 340
кэш-память 100, 118
— дисковая 100
кэш-строка 119
Латентность 96, 271
ленд 170
линк 296
Маршрутизация 283
маска
— прерывания 92
массив
— волновой 233
— дисковый 161
— систолический 235, 317
масштабируемость ЭВМ или ВС 50
МВС-100 241, 298
МВС-1000 241, 298, 300
метод
— быстрого реагирования 88
— доступа к памяти 95
— обратной записи 124
— повышения быстродействия памяти 105
— помеченного оператора (опорных точек) 88
— сквозной записи 124
— структурный повышения быстродействия памяти 109
механизм преобразования адресов 119
микропрограмма 60
многопроцессорная вычислительная система 7
модель параллельного программирования 243
модуль
— ввода-вывода 133
— вычислительный 269, 298
— памяти 109
монитор 200
мышь 185
мэйнфрейм 11, 21
Надежность ЭВМ или ВС 40

накопитель
— на гибких магнитных дисках 160
— на жестких магнитных дисках 156
— на магнитной ленте 179
— на магнитооптических дисках 160
— на оптических дисках 170
нанотехнология 323
нанотрубка 326
наработка на отказ 45
насыщение системы прерывания 87
нейрокомпьютер 348
нейрон 350
Обработчик прерывания 84, 90
общая шина 136
опрос 137
организация
— ввода-вывода
— канальная 136, 140
— шинная 136, 143
— памяти блочная 109
— параллельной обработки 220
отказ 42
отказоустойчивость ЭВМ или ВС 43
оценка стоимости ПО 52
Память
— ассоциативная 96, 116, 267, 314
— виртуальная 97, 126
— вторичная 127
— данных 120
— оптическая 335
— основная (оперативная) 97
— первичная 126
— тегов 120
— физическая 97
параграф 79
парадигма параллельного программирования 242
параллелизм
— данных 242
— задач 242, 254
— скрытый 310
параллельная
— архитектура 232
— обработка 220
— программа 241
перестановочная сеть 268
переход
— безусловный 71
— вызов процедуры 73
— условный 72
персептрон 353, 357
пиксел 186
пит 170
показатель качества 28
поле 79
польская запись 67
попадание 98
потоковая ВС 233, 312
прерывание 91, 138
прибор с зарядовой связью (ПЗС) 188
принтер
— лазерный 197
— линейно-матричный 195
— матричный 194
— струйный 196
— термический 200
принцип
— адресности 58
— двоичного кодирования 57
— иерархический организации памяти 97
— координатный адресации ячеек памяти 102
— локальности 98
— однородности памяти 58
— программного управления 58, 348
принцип фон Неймана 57
приоритет прерывания 89
программа прерывающая 91
программный счетчик 57, 59, 86, 309
производительность 30

— коммуникационной среды 209
— пиковая 32
— реальная 33
промах 98, 124
пропускная способность 271
протокол
— DASH 294
— MESI 291
профилирование 257
процессор
— ассоциативный 267
— вычислительный 298
— графический 214
— звуковой 217
— оптический 333
— потоковый 316
— связной 298
— управляющий 263
— элементарный 263, 352
Распределение
— адресов 111
— множественно-ассоциативное 123
— полностью ассоциативное 121
— прямое 119
расслоение памяти 111
регистр
— адреса памяти 111
— аккумулятор 68
— векторный 262
— общего назначения (РОН) 68
редукционные ВС 233
ремонтопригодность ЭВМ
или ВС 44
рендеринг 213
решающее поле 269, 296
Сбой 42
световое перо 186
сегмент 79, 127
сектор 156
сервер 11, 18
— сервер-лезвие 21
система
— вычислительная 7
— многопроцессорная 7
— команд 63
— обработки данных (СОД) 6
— обработки информации (СОИ) 7
— прерываний 81
систолизация 319
сканер 187
слово 79
слово состояния программы
(ССП) 86, 91
совместимость и мобильность ПО 51
способ адресации 60, 62, 73, 78
стандарт
— AC-97 218
— MPI 255
— OpenMP 253
— PVM 254
— спецификация 99 12
— шин 149
стек 67, 125
стекер 179
столбец 103
страница 79
стратегия
— выборки 119
— замещения 119, 124
— обновления ОП 119
— размещения 119
стример 179
строка 103
структурирование буферного пула 284
схема параллелизма 242, 310
счетчик команд 57
Таблица
— векторов прерываний 91
— страниц 128
тег 120
текстурирование 213
тип прерывания 84

точность информации 43
транзакция 144
транспьютер 238, 297
Указатель команды 57
управление
— вводом-выводом 136
— вычислительными ресурсами 304
— коммутаторами 283
уровень прерывания 91
ускорение счета 224
устройство
— арифметико-логическое 57
— ввода 57
— вывода 57
— запоминающее 95
— ассоциативное 116, 266
— внешнее 126
— оперативное (ОЗУ) 101
— постоянное (ПЗУ) 101
— сверхоперативное (СОЗУ) 100
— управления 57
Фирма-производитель
— Acorn Computers 18
— AMD 16
— Amdahl 22
— Analog Devices 301
— Apple 17
— Burroughs 68
— Casio 17
— CDC 236
— Compaq 13, 18
— Cray Research 24, 237
— Creative 218
— DEC 11, 31
— Etnus 257
— Fujitsu 288, 347
— Hewlett Packard (HP) 68
— Hitachi 21
— IBM 21, 22
— Inmos 68
— Intel 12,
— Matrox 215
— Microsoft 18
— NEC 13,
— Nothern Telecom 155
— Palm Computing 18
— SGI 12,
— Sun Microsystems 12
— Texas Instruments 67
— Toshiba 175
— Trensmeta 16
— U.S. Robotics 18
— VIA 16
флэш-память 175
формат команды 73
фотоэлектронный умножитель 188
фрагментация памяти 126
функция
— активации 352
— готовности 46, 47
Цветовая температура 206
цена обмена 271
центральная часть ЭВМ 57
цикл
— графа межмодульных связей 280
— команды 61
— памяти 100, 96
цилиндр 156
Частота отказов 45
чередование адресов 111
Шейдер 214
шина
— ATA 153
— EISA 144, 151
— IDE 153, 158
— ISA 144
— PCI 151, 152
— SAS 150, 154
— SATA 150, 153, 154
— SCSI 154
— USB 149, 155, 158
— VME 152

— асинхронная 145
— локальная 144
— общая 143
— синхронная 145
— системная 144
шпиндель 156
ЭВМ 6
— криогенная 343
— молекулярная 327
— настольная 11
— персональная ЭВМ 11
— рабочая станция 11
— портативная 11, 13
— карманный ПК 13, 17
— ноутбук 13
— последовательного типа 57
— супер-ЭВМ 11, 23
— управляющая 296
— эталонная 34
экономичность ЭВМ или ВС 52
электронно-лучевая трубка 201
элемент
— запоминающий (ЗЭ) 102
— процессорный (ПЭ) 263, 317
Эльбрус 240
энергонезависимость 97
эффективность ЭВМ
или ВС 29
Ядро
— микросхемы памяти 103, 105
ярусно-параллельная форма
программы 222

Список сокращений

Сокращение	*Расшифровка*
ARM	Advanced RISC Machines
CAS	Column-Access Strobe
CD	Compact Disk
CISC	Complex Instruction Set Computer
CM	систолический массив
CRT	Cathode Ray Tube
DRAM	Dynamic RAM
DTR	Desktop Replacement
DVD	Digital Versatile Disk
FDD	Floppy Disk Drive
FIFO	First Input First Output
HDD	Hard Disk Drive
LCD	Liquid Crystal Display
LIFO	Last In First Out
LRU	Least Recently Used
MFLOPS	Million FLoating point Operations Per Second
MIMD	Multiple Instruction stream / Multiple Data stream
MIPS	Million Instructions Per Second
MISD	Multiple Instruction stream / Single Data stream
MPP	Massively Parallel Processing
NUMA	Non-Uniform Memory Access
PCM	Pulse Code Modulation
PIO	Processor Input-Output
RAID	Redundant Array of Independent Disks
RAIT	Redundant Array of Independent Tapes
RAM	Random Access Memory
RAS	Row-Access Strobe
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
SIMD	Single Instruction stream / Multiple Data stream

SISD	Single Instruction stream / Single Data stream
SMP	Symmetrical Multiprocessor
SPMD	Single Program — Multiple Data
SRAM	Static RAM
VLIW	Very Long Instruction Word
АЗУ	ассоциативное запоминающее устройство
АЛУ	арифметико-логическое устройство
АП	ассоциативная память
АСОИ	автоматическая система обработки информации
АЦП	аналого-цифровой преобразователь
ВЗУ	внешнее запоминающее устройство
ВК	вычислительный комплекс
ВМ	вычислительная машина вычислительный модуль
ВП	внешняя память виртуальная память
ВС	вычислительная система
ГЗ	граф зависимостей
ГПС	граф потока сигналов
ЖК	жидкокристаллический
ЗМ	запоминающий массив
ЗУ	запоминающее устройство
ЗЭ	запоминающий элемент
ИВК	информационно-вычислительный комплекс
ИВС	информационно-вычислительная система
ИКМ	импульсно-кодовая модуляция
ИЛС	информационно-логическая система
ИМС	интегральная микросхема
ИНС	искусственная нейронная сеть
КВ	квантовые вычисления
КВВ	канал ввода-вывода
КК	квантовый компьютер

КОП	код операции
КПДП	контроллер прямого доступа к памяти
КПК	карманный персональный компьютер
МВВ	модуль ввода-вывода
МВС	много(ая вычислительная машина
МКМД	множественный поток команд — множественный поток данных
МКОД	множественный поток команд — одиночный поток данных
МОД (МО-диск)	магнитооптический диск
НГМД	накопитель на гибком магнитном диске
НЖМД	накопитель на жестком магнитном диске
НМЛ	накопитель на магнитной ленте
НОД	накопитель на оптическом диске
ОЗУ	оперативное запоминающее устройство
ОКМД	одиночный поток команд — множественный поток данных
ОКОД	одиночный поток команд — одиночный поток данных
ОП	основная (оперативная) память
ОС	операционная система (среда)
ОУ	обрабатывающее устройство
ПДП	прямой доступ к памяти
ПЗС	прибор с зарядовой связью
ПЗУ	постоянное запоминающее устройство
ПК	персональный компьютер
ПЛИС	программируемая логическая интегральная схема
ПО	программное обеспечение
ПЭ	показатель эффективности
ПЭВМ	персональная ЭВМ
РС	регистр состояния
РУ	регистр управления
СА	способ адресации
СБИС	сверхбольшая интегральная схема
СОД	система обработки данных

СОЗУ	сверхоперативное запоминающее устройство
СОИ	система обработки информации
СП	сверхпроводимость
ССП	слово состояния процессора (программы)
УВВ	устройство ввода-вывода
УУ	устройство управления
ФЭУ	фотоэлектронный умножитель
ЦАП	цифро-аналоговый преобразователь
ЦП	центральный процессор
ЭВМ	электронная вычислительная машина
ЭЛТ	электронно-лучевая трубка
ЯВУ	язык высокого уровня
ЯПФ	ярусно-параллельная форма

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3

ГЛАВА 1. БАЗОВЫЕ СВЕДЕНИЯ ТЕОРИИ ВЫЧИСЛИТЕЛЬНЫХ МАШИН, КОМПЛЕКСОВ И СИСТЕМ 6

1.1. ТЕРМИНОЛОГИЯ В ОБЛАСТИ ЭВМ, ВС И КОМПЛЕКСОВ 6

1.2. КЛАССИФИКАЦИЯ ЭВМ И ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ 10

1.2.1. Классификация ЭВМ 10

1.2.1.1. *Настольные ЭВМ (персональные ЭВМ и рабочие станции)* 11

1.2.1.2. *Портативные ЭВМ* 13

1.2.1.3. *Серверы* 18

1.2.1.4. *Мэйнфреймы* 21

1.2.1.5. *Супер-ЭВМ* 23

1.2.2. Классификация вычислительных систем 24

1.3. ПОКАЗАТЕЛИ КАЧЕСТВА ФУНКЦИОНИРОВАНИЯ ЭВМ И ВС 28

1.4. ЭФФЕКТИВНОСТЬ ЭВМ И ВС 29

1.5. ПРОИЗВОДИТЕЛЬНОСТЬ ЭВМ И ВС 30

1.5.1. Понятие производительности 30

1.5.2. Пиковая производительность 32

1.5.3. Реальная производительность 33

1.5.4. Время ответа ЭВМ и ВС 40

1.6. ПОКАЗАТЕЛИ НАДЕЖНОСТИ И ГОТОВНОСТИ ЭВМ И ВС 40

1.6.1. Основные понятия надежности ЭВМ и ВС 40

1.6.2. Количественные показатели надежности, готовности и ремонтопригодности технических средств ЭВМ и ВС 44

1.6.3. Показатели надежности программных средств ЭВМ и ВС 47

1.6.4. Методы и средства повышения надежности ЭВМ и ВС 48

1.7. АДАПТИВНОСТЬ ЭВМ И ВС 50

1.8. ЭКОНОМИЧНОСТЬ ЭВМ И ВС 52

Вопросы для самоконтроля 55

ГЛАВА 2. ЭВМ ПОСЛЕДОВАТЕЛЬНОГО ТИПА 57

2.1. СТРУКТУРА, ОСНОВНЫЕ КОМПОНЕНТЫ И ПРИНЦИПЫ ФУНКЦИОНИРОВАНИЯ ПОСЛЕДОВАТЕЛЬНЫХ ЭВМ 57

2.2. ЦИКЛ КОМАНДЫ 61
2.3. СИСТЕМА КОМАНД ЭВМ 63
2.3.1. Классификация по составу и сложности команд 63
2.3.2. Классификация по месту хранения операндов 67
2.3.3. Классификация по способу обработки данных и типу операндов 70
2.4. ФОРМАТЫ КОМАНД 73
2.5. СПОСОБЫ АДРЕСАЦИИ 77
2.6. СИСТЕМА ПРЕРЫВАНИЙ ПРОГРАММ ЭВМ 81
2.6.1. Необходимость системы прерываний 81
2.6.2. Общая организация системы прерываний 84
2.6.3. Определение момента прерывания и приоритета запроса 88
2.6.4. Передача управления прерывающей программы и возврат из нее 90
Вопросы для самоконтроля 93
ГЛАВА 3. АРХИТЕКТУРА ПАМЯТИ ЭВМ 95
3.1. ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ПАМЯТИ ЭВМ 95
3.2. ИЕРАРХИЧЕСКАЯ ОРГАНИЗАЦИЯ ПАМЯТИ ЭВМ 97
3.3. АРХИТЕКТУРА ОСНОВНОЙ ПАМЯТИ 101
3.3.1. Принципы построения оперативных запоминающих устройств 101
3.3.2. Методы повышения быстродействия основной памяти 105
3.3.2.1. *Оптимизация доступа к ОЗУ* 106
3.3.2.2. *Структурные методы повышения быстродействия ОП* 109
3.4. ПАМЯТЬ С АССОЦИАТИВНЫМ ДОСТУПОМ 116
3.5. КЭШ-ПАМЯТЬ 118
3.5.1. Стратегии размещения 119
3.5.1.1. *Прямое распределение* 119
3.5.1.2. *Полностью ассоциативное распределение* 121
3.5.1.3. *Множественно-ассоциативное распределение* 123
3.5.2. Стратегии обновления основной памяти и стратегии замещения 124
3.6. АРХИТЕКТУРА ВИРТУАЛЬНОЙ ПАМЯТИ 125

3.6.1. Страничная организация памяти 127
3.6.2. Сегментация памяти 130
Вопросы для самоконтроля 132

ГЛАВА 4. ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА 133

4.1. ОБЩАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ ВВОДА-ВЫВОДА 133
4.1.1. Структура и функции модуля ввода-вывода 133
4.1.2. Методы управления вводом-выводом 136
4.2. КАНАЛЬНАЯ ОРГАНИЗАЦИЯ ПОДСИСТЕМ ВВОДА-ВЫВОДА 140
4.3. ШИННАЯ ОРГАНИЗАЦИЯ ПОДСИСТЕМЫ ВВОДА-ВЫВОДА 143
4.3.1. Общая организация шин 143
4.3.2. Алгоритмы арбитража 146
4.4. СТАНДАРТЫ ШИН 149
4.4.1. Общие тенденции в развитии шин 149
4.4.2. Шины «большого» интерфейса 150
4.4.3. Шины «малого» интерфейса 153
4.5. УСТРОЙСТВА ВВОДА-ВЫВОДА 155
4.5.1. Оосновные типы устройства ввода-вывода 155
4.5.2. Запоминающие устройства 156
4.5.2.1. *Накопители на жестких магнитных дисках* 156
4.5.2.2. *Накопители на гибких магнитных дисках и магнитооптические накопители* 160
4.5.2.3. *Дисковые массивы и уровни RAID* 161
4.5.2.4. *Оптические диски* 170
4.5.2.5. *Флэш-память* 170
4.5.2.6. *Устройства резервного копирования информации* 178
4.5.3. Устройства ввода информации 182
4.5.3.1. *Клавиатура* 182
4.5.3.2. *Манипуляторы «мышь», джойстик, световое перо* 185
4.5.3.3. *Сканеры* 187
4.5.3.4. *Трехмерные сканеры* 193
4.5.4. Устройства вывода информации 194
4.5.4.1. *Матричные принтеры* 194
4.5.4.2. *Струйные принтеры* 196

4.5.4.3. *Лазерные принтеры* 197
4.5.4.4. *Термические принтеры* 200
4.5.4.5. *Мониторы (дисплеи)* 200
4.5.4.6. *Видеоадаптеры* 208
4.5.4.7. *Звуковые адаптеры* 215
Вопросы для самоконтроля 218

ГЛАВА 5. ПАРАЛЛЕЛЬНЫЕ ЭВМ И ВС 220

5.1. ТИПЫ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ИНФОРМАЦИИ....220
5.2. КЛАССИФИКАЦИЯ ПАРАЛЛЕЛЬНЫХ ВС 226
5.2.1. Признаки классификации параллельных ВС....226
5.2.2. Классификация Флинна 227
5.2.3. Классификация Фенга....229
5.2.4. Классификация Дункана. Понятие о нетрадиционных параллельных архитектурах....231
5.2.5. Современная классификация параллельных ВС 233
5.3. ИСТОРИЯ РАЗВИТИЯ ПАРАЛЛЕЛИЗМА В АРХИТЕКТУРЕ ЭВМ...235
5.4. ПОНЯТИЕ О ПАРАЛЛЕЛЬНОМ ПРОГРАММИРОВАНИИ....241
5.4.1. Парадигмы параллельного программирования 241
5.4.2. Модели параллельного программирования 243
5.4.3. Понятие о векторизации программ....246
5.4.4. Средства параллельного программирования....251
5.4.4.1. *Языковые средства* 251
5.4.4.2. *Коммуникационные библиотеки* 254
5.4.4.3. *Средства отладки, мониторинга и оптимизации параллельных программ*....256
5.5. КЛАСС MISD— КОНВЕЙЕРНАЯ ОБРАБОТКА ИНФОРМАЦИИ 258
5.6. СИСТЕМЫ КЛАССА SIMD 260
5.6.1. Векторные ВС 260
5.6.2. Матричные системы....263
5.6.3. Ассоциативные системы 266
5.7. СИСТЕМЫ КЛАССА MIMD....269
5.7.1. Обобщенная структура MIMD-систем 269
5.7.2. Коммуникационные среды MIMD-систем 270

5.7.2.1. *Основные характеристики коммуникационных сред* 270
5.7.2.2. *Простые коммутаторы* 272
5.7.2.3. *Составные коммутаторы* 274
5.7.2.4. *Коммутатор Клоза* 277
5.7.2.5. *Баньян-сети* 278
5.7.2.6. *Распределенные составные коммутаторы* 280
5.7.2.7. *Управление коммутаторами* 283
5.7.3. MIMD-системы с общей памятью 286
5.7.3.1. *Архитектура MIMD-систем с общей памятью* 286
5.7.3.2. *Обеспечение когерентности памяти в ВС с общей памятью* 289
5.7.3.3. *Протокол MESI* 291
5.7.3.4. *Протоколы поддержки когерентности для распределенной общей памяти* 294
5.7.4. Массово-параллельные ВС 296
5.7.4.1. *Общие принципы построения MPP-систем* 296
5.7.4.2. *Отечественные массово-параллельные ВС МВС-100х* 298
5.7.5. Кластерные ВС 301
5.8. УПРАВЛЕНИЕ ВЫЧИСЛИТЕЛЬНЫМИ РЕСУРСАМИ В ПАРАЛЛЕЛЬНЫХ ВС 304
5.9. ПАРАЛЛЕЛЬНЫЕ ВС С НЕТРАДИЦИОННОЙ АРХИТЕКТУРОЙ 309
5.9.1. Вычислительные системы, управляемые потоком данных 309
5.9.2. Систолические массивы 317
Вопросы для самоконтроля 320
ГЛАВА 6. ПЕРСПЕКТИВНЫЕ НАПРАВЛЕНИЯ В РАЗВИТИИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ 323
6.1. НАНОТЕХНОЛОГИЯ 323
6.2. МОЛЕКУЛЯРНЫЕ КОМПЬЮТЕРЫ 327
6.3. ОПТИЧЕСКИЙ КОМПЬЮТЕР 332
6.4. КВАНТОВЫЕ КОМПЬЮТЕРЫ 338
6.5. КРИОГЕННЫЕ ЭВМ 343
6.6. НЕЙРОКОМПЬЮТЕРЫ 348
6.6.1. Общие положения 348

6.6.2. Нейронная сеть человека 350
6.6.3. Простейшие нейрокомпьютеры 352
6.6.4. Архитектура нейронной сети 353
6.6.4.1. *Обучение* 353
6.6.4.2. *Модели сетей* 356
6.6.5. Промышленные нейрокомпьютеры и перспективы их развития 358
Вопросы для самоконтроля 360
Литература 361
Предметный указатель 367
Список сокращений 374

Учебное издание

Тихонов Виктор Алексеевич
Баранов Антон Викторович
Организация ЭВМ и систем

Заведующая редакцией *Т. А. Денисова*
Корректор *Л. М. Лисицына*
Компьютерная верстка *С. Н. Авилкина*

Издательство «Гелиос АРВ»
Издательская лицензия ЛР №066255
Гигиенический сертификат № 77.99.02.953.Д000390.01.06
Юр. адрес: 107140, г. Москва, Верхняя Красносельская ул., 16
Факт. адрес: 125438, г. Москва, 4-й Лихачевский пер., д. 15, оф. 615
Тел./факс: (495) 156-71-65, e-mail: info@gelios-arv.ru
Адрес в Internet: http://www.gelios-arv.ru

Формат 70x100/16. Печать офсетная. 24 п. л.

Тираж 1000 экз. Бумага офсетная.

Заказ № 4

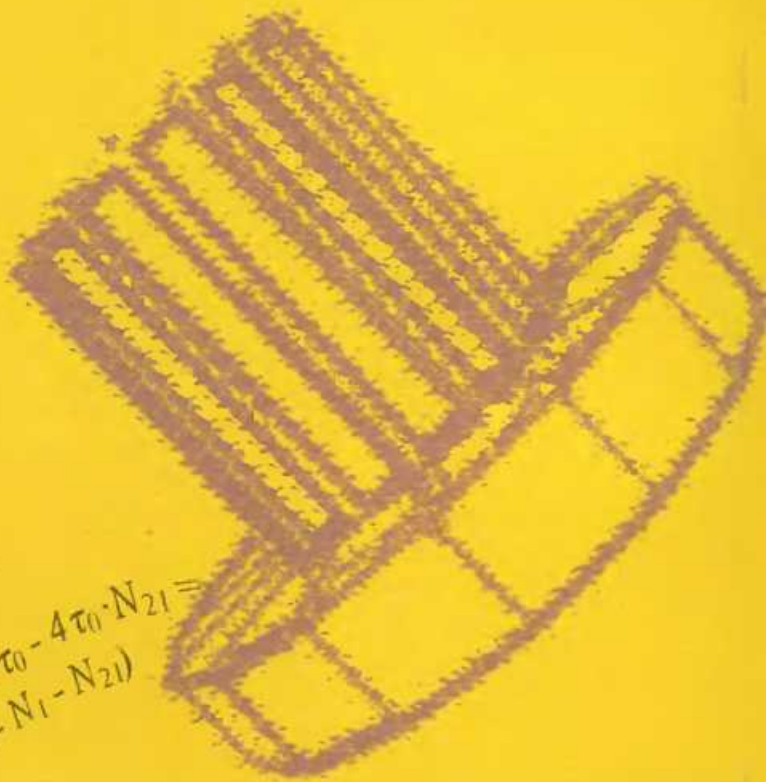
Отпечатано с готовых диапозитивов
в типографии ГП «Облиздат».
248640, г. Калуга, пл. Старый торг, 5

252-45

УЧЕБНИК

Учебник посвящен систематическому изложению принципов построения, архитектуре и компонентам современных ЭВМ и вычислительных систем. Соответствует программе курса «Организация ЭВМ и систем».

Для студентов, обучающихся по специальности «Вычислительные машины, системы, комплексы и сети» (230101), может быть полезен и студентам других специальностей, где имеются соответствующие дисциплины, а также специалистам в области ЭВМ и вычислительных систем.



В. А. Тихонов, А. В. Баранов

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ